Complexity Reduction in Information Systems Modelling

Anne Helga Seltveit

Information Systems Group
Faculty of Electrical Engineering and Computer Science
The Norwegian Institute of Technology
The University of Trondheim

December 13, 1994
Abstract

An approach to information systems development based on systematic complexity reduction is presented. It facilitates integrated specifications and at the same time it provides facilities that allow the developers to split a complicated task into a set of more comprehensible subtasks. Integrated specifications are facilitated through a set of structuring mechanisms. A coupling between process based and rule based approaches is formally defined and a set of links to relate specifications at different abstraction levels are suggested. By providing the notion of viewspecs and filters to generate the viewspecs, details can be separated in a systematical manner, enabling developers to concentrate on relevant parts of the system at a time.

Through the introduction of viewspecs the suggested approach recognises the need to adopt multiple perspectives when developing large and complex information systems. The possibility of using pre-defined filters, user-defined filters and combinations of these, provides the developer using the complexity reduction approach with a great deal of freedom in focusing on different aspects of a system. Rather than operating on a full specification, relevant viewspecs can be applied at different stages of the development process, e.g., in rule capture. The viewspecs are used 'actively' during the development in the sense that they do not only serve as passive components or projections of a set of specifications (e.g., for presentation purposes) but also serve as a basis for entering new information into specifications. This is achieved by allowing viewspecs to be updated and then eventually integrated into a new version of the full specification. Moreover, multiple viewspecs are allowed to co-exist. To glue the contradictory actions of integration and fragmentation, additional support is provided to allow the developer to go back and forth between the integrated specifications and their associated viewspecs.

The suggested approach is developed in the context of the PPP and TEMPORA projects. The thesis focuses on how the TEMPORA and PPP approaches can be extended with complexity reducing features to deal with development of large and complex systems. The problems associated with incorporation of rules in information systems development are also addressed.
## Contents

Preface .................................................. vii

1 Introduction ........................................ 1
   1.1 Information Systems Engineering ................. 1
   1.2 Tame and Wicked Problems .......................... 2
   1.3 Taming Problems Requires Understanding .......... 2
   1.4 The Idea of Complexity Reduction ................ 5
   1.5 Case Studies ...................................... 10
   1.6 Examples of Application of Complexity Reduction .. 15

2 State-of-the-Art Survey .............................. 23
   2.1 Classification of Complexity Reducing Facilities .... 23
   2.2 Conceptual Modelling Approaches and Complexity ..... 28
   2.3 Complexity Reducing Facilities in ICASE Tool Environments .. 56
   2.4 CASE Tools ...................................... 58
   2.5 Chapter Summary .................................. 65

3 Baseline .............................................. 67
   3.1 An Overview of the TEMPORA Approach .............. 67
   3.2 The TEMPORA Languages ............................ 68
   3.3 Experiences Using the TEMPORA Approach .......... 75
   3.4 Chapter Summary .................................. 81

4 Requirements ......................................... 83
   4.1 Classification of Requirements ....................... 83
   4.2 Language Dependent Requirements .................... 85
   4.3 Method and Tool Dependent Requirements .......... 89
   4.4 Chapter Summary .................................. 91

5 Combining Process Based and Rule Based Approaches .... 93
   5.1 The Conceptual Level Coupling ....................... 94
   5.2 The Design Level Coupling ........................ 101
   5.3 Rule Execution in a Temporal Database Environment .. 105
   5.4 Chapter Summary .................................. 108

6 Structuring Mechanisms ................................ 109
   6.1 A Framework for Structuring Mechanisms ............ 109
   6.2 Structuring Mechanisms in TEMPORA - An Overview ... 114
   6.3 Intra-Language Links ................................ 114
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
<td>Inter-Language Links</td>
<td>120</td>
</tr>
<tr>
<td>6.5</td>
<td>Inter-Level Links</td>
<td>125</td>
</tr>
<tr>
<td>6.6</td>
<td>Chapter Summary</td>
<td>129</td>
</tr>
<tr>
<td>7</td>
<td>Filtering Mechanisms</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Framework for Filtering Mechanisms</td>
<td>131</td>
</tr>
<tr>
<td>7.2</td>
<td>An Overview of Filters in TEMPORA</td>
<td>138</td>
</tr>
<tr>
<td>7.3</td>
<td>PID Filters</td>
<td>139</td>
</tr>
<tr>
<td>7.4</td>
<td>ERT Filters</td>
<td>156</td>
</tr>
<tr>
<td>7.5</td>
<td>ERL Filters</td>
<td>168</td>
</tr>
<tr>
<td>7.6</td>
<td>Chapter Summary</td>
<td>171</td>
</tr>
<tr>
<td>8</td>
<td>The Complexity Reduction Approach</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>An Overview of the Approach</td>
<td>173</td>
</tr>
<tr>
<td>8.2</td>
<td>Filtering</td>
<td>178</td>
</tr>
<tr>
<td>8.3</td>
<td>Presentation</td>
<td>191</td>
</tr>
<tr>
<td>8.4</td>
<td>Entering New Specification Details</td>
<td>201</td>
</tr>
<tr>
<td>8.5</td>
<td>Inclusion of Changes</td>
<td>205</td>
</tr>
<tr>
<td>8.6</td>
<td>Complexity reduction in the TEMPORA Development Process</td>
<td>211</td>
</tr>
<tr>
<td>8.7</td>
<td>Chapter Summary</td>
<td>211</td>
</tr>
<tr>
<td>9</td>
<td>Applying Complexity Reduction in PPP</td>
<td></td>
</tr>
<tr>
<td>9.1</td>
<td>The PPP Approach</td>
<td>213</td>
</tr>
<tr>
<td>9.2</td>
<td>Complexity Reducing Features in PPP</td>
<td>219</td>
</tr>
<tr>
<td>9.3</td>
<td>Incorporating Rules in PPP</td>
<td>222</td>
</tr>
<tr>
<td>9.4</td>
<td>The Viewing Facility</td>
<td>226</td>
</tr>
<tr>
<td>9.5</td>
<td>Chapter Summary</td>
<td>236</td>
</tr>
<tr>
<td>10</td>
<td>Evaluation of the Complexity Reduction Approach</td>
<td>237</td>
</tr>
<tr>
<td>10.1</td>
<td>Evaluation with Respect to Stated Requirements</td>
<td>237</td>
</tr>
<tr>
<td>10.2</td>
<td>Evaluation with Respect to Other Approaches</td>
<td>246</td>
</tr>
<tr>
<td>10.3</td>
<td>Experiences using the Suggested Approach</td>
<td>251</td>
</tr>
<tr>
<td>10.4</td>
<td>General Applicability</td>
<td>256</td>
</tr>
<tr>
<td>10.5</td>
<td>Chapter Summary</td>
<td>258</td>
</tr>
<tr>
<td>11</td>
<td>Concluding Remarks and Future Directions</td>
<td></td>
</tr>
<tr>
<td>11.1</td>
<td>Major Contributions</td>
<td>261</td>
</tr>
<tr>
<td>11.2</td>
<td>Limitations and Future Directions</td>
<td>263</td>
</tr>
<tr>
<td>A</td>
<td>Rule Relations</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Versioning and Configuration Management</td>
<td>269</td>
</tr>
<tr>
<td>C</td>
<td>PPP Repository Structure</td>
<td>275</td>
</tr>
<tr>
<td>D</td>
<td>Example of the Use of Rules in the PPM</td>
<td>277</td>
</tr>
<tr>
<td>E</td>
<td>Rules - Sweden Post Case Study</td>
<td>283</td>
</tr>
<tr>
<td>F</td>
<td>Definition of Filters</td>
<td>297</td>
</tr>
<tr>
<td>CONTENTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>List of Figures</td>
<td>301</td>
<td></td>
</tr>
<tr>
<td>List of Tables</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>Bibliography</td>
<td>309</td>
<td></td>
</tr>
</tbody>
</table>
Problem Statement

A specification of an information system plays a dual role in the systems development process: it serves as a basis for communication and understanding as well as a basis for the coding of software. This implies that a specification must contain massive amounts of details expressed in a formal way and at the same time it must be understandable for the actors involved in the development process. The problem addressed in this thesis is formulated as:

Is it possible to find a balance between the contradictory roles of a specification of an information system and if so, can an appropriate information systems modelling process be found?

In this thesis an ‘information system’ is conceived to consist of parts that may be fully automated as well as parts that may involve human participation. A ‘specification’ (also called model object as well as design object) refers to the result of work activity in the modelling process. The term ‘information systems modelling process’ includes method support as well as tool support. The ‘contradictory roles of a specification’ is considered in the context of large scale information systems development. In particular, this thesis addresses two issues where contemporary modelling approaches have not yet provided satisfactory solutions:

- **Fragmentation of specifications.** Many modelling approaches and several specification languages are used in various development phases, from requirements development to coding. The state-of-the-art is that models and languages are not integrated over phase-boundaries. There is a need for consolidation of the various approaches.

- **Complexity explosion.** As more details are added during the development process, a complexity ‘explosion’ may occur. Contemporary modelling approaches suffer from the lack of support to deal with this information overload. There is a need for a systematical way to deal with large number of details in specifications, as well as during the modelling process.
The baseline of this work is the data based and rule based approaches which have been developed in the ESPRIT project TEMPORA, and the process based approach PPP which is being developed in the Information Systems Group at NTH.

**Objectives**

The objectives of the research are:

- *To find ways to deal with complexity of specifications.* It should suggest how large and complex specifications can be dealt with to overcome the problems of fragmentation and complexity explosion.

- *To find ways of delivering comprehensive support during the development process.* To facilitate development of large and complex specifications, it should also provide active support during the modelling process.

Both objectives focus on how to achieve the dual purpose of a specification better than what is the current situation in the contemporary approaches.

**Approach**

The thesis addresses central issues of process based and rule based approaches when used for large-scale information systems development. It identifies major problems of process based and rule based approaches and proposes a way to combine the two approaches. The resulting modelling approach can describe both static\(^1\) and dynamic aspects including the temporal dimension. To deal with massive amounts of details in specifications as well as in the development process, an approach based on systematic complexity reduction is suggested. The following issues are particularly addressed in the thesis:

- *Integrated specifications.* The use of more than one language complicates the development process by producing heterogenous specifications. To relate such specifications at different levels of details, a set of structuring mechanisms are proposed.

- *Systematic complexity reduction.* Concepts and mechanisms to facilitate development of large information systems based on systematic complexity reduction are proposed.

\(^1\)The data based approach is however, less emphasised in this work because its coupling to the process based and rule based approaches is already well understood.
• *Method and tool support.* Facilities for comprehensive method and tool support are discussed.

The best way to investigate if a suggested approach works is to use it. This presumes that a tool environment exists. Unfortunately, this is not the case for our proposal. A massive effort is needed to implement a full-fledged CASE tool. This goes beyond what can be achieved within the scope of a dr. ing. thesis. Therefore, the validation of the suggested approach in this thesis is based on limited experiments where only the major features of the approach have been used in modelling industrial-sized problems. The approach is also compared with contemporary approaches and the general applicability of the suggested approach is discussed.

Central issues of process based and rule based approaches to large-scale information systems development are identified based on a literature study, on the author’s own experience, on case studies and examples using the PPP and TEMPORA approaches and on otherwise available information about contemporary approaches, e.g., demonstrations of CASE tools and experience gained using the tools.

The elaboration of aspects of the process and rule based approaches to large-scale information systems development to a similar level of detail is beyond the scope of this thesis. We have to limit ourselves to something more realistic:

• We concentrate on methodological issues.

• The focus is on the early phases of development, i.e., the conceptual level.

• Rather than focusing on knowledge acquisition techniques as such, we concentrate on how we can elaborate our modelling approach (including language features, method support, and tool support) to improve the process of capturing relevant information by exploiting the combination of diagrammatic and non-diagrammatic languages.

• Details about design and run-time related issues are only included when necessary to illuminate central issues of the approach.

**Major Results Achieved**

The objectives of the thesis are to find ways to deal with complexity in specifications and to find ways to support the modelling process. They have been met as follows:

• Concerning how to deal with complexity of specifications, the major contributions are:
  
  – Framework for complexity reduction in information systems development.
    
  This includes:
An understanding of the concept of complexity reduction. Few contemporary approaches have addressed the duality of specifications. A better understanding of complexity in specifications and ways to reduce the complexity are therefore important and useful.

A framework for structuring a conceptual model is proposed. This gives us the means to describe how information should be structured within and across specification levels. This is crucial in order to reduce fragmentation in specifications.

A framework for filtering mechanisms is proposed. This provides us with the means to focus on essential issues by suppressing irrelevant details.

- Complexity reducing facilities for the TEMPORA and PPP languages. This includes:
  - The TEMPORA and PPP structuring mechanisms are further elaborated.
  - Filtering mechanisms for the TEMPORA and PPP languages are suggested. In particular, a set of filters are formally defined. The filters are used to generate more comprehensible views of specifications by suppressing irrelevant details.
  - Suggestions for how to deal with typical problems of rule based approaches in information systems development, and in particular how these can be dealt with in PPP and TEMPORA.

- Concerning support during the development process, the major contributions are:
  - Specification of comprehensive method support. This specifies how complexity reducing facilities can be applied in the modelling process.
  - Specification of computerised support for an approach based on complexity reduction. A discussion and suggestion for how complexity reducing facilities can be implemented in the PPP CASE tool, are provided. This is used as a basis for the implementation of such facilities in the prototype which is being developed.

### Outline of the Thesis

The thesis is structured as follows:

The idea of complexity reduction in information systems modelling is briefly described and examples which are used throughout the thesis are presented (Chapter 1). Major trends in Information Systems Engineering are outlined and a state-of-the-art report on how contemporary approaches to conceptual modelling deals
with complexity, are given (Chapter 2). Then the TEMPORA approach is presented together with experiences from using the approach in case studies and on examples (Chapter 3).

The overall requirements to an approach to large-scale information systems development based on systematic complexity reduction are identified (Chapter 4). A coupling between process and rule based approaches is defined (Chapter 5) and a framework for integrating different parts of TEMPORA specifications is described (Chapter 6). A set of complexity reducing mechanisms to generate different views of a specification is defined (Chapter 7). The process of building up specifications based on views is examined and mechanisms to support such a process are proposed (Chapter 8).

The feasibility of the approach is illustrated as follows: In addition to outline how some of the suggested features are implemented in the TEMPORA CASE tool (Chapters 5 to 8), we suggest how the PPP CASE tool can be extended to deal with rules and to encompass complexity reducing features (Chapter 9). An overview of the implementation effort is presented. An evaluation with respect to stated requirements and a comparison with other approaches are provided (Chapter 10). It also reports experiences from case studies and discusses the general applicability of the suggested approach. Finally, some concluding remarks are given (Chapter 11).

About the Work

The work reported has partly taken place in the context of the ESPRIT II Project number E2469 TEMPORA and partly in the context of the RHAPSODY project of the Information Systems Group at NTH.

The TEMPORA project was funded by the Commission of the European Communities under the ESPRIT R&D programme. The partners in the TEMPORA consortium are: BIM (Belgium), Hitec (Greece), Imperial College (UK), Logic Programming Associates (UK), NTH, through SINTEF (Norway), SISU (Sweden), University of Liege (Belgium), and UMIST (UK). The EFTA partners SINTEF and SISU are funded by their national research councils NTFN and STU, respectively.

During the working period, the author has collaborated with a number of people. To make sure that the thesis cites the source for anything that is not original with the author, we adopt the following conventions:

1. Work done by others: the source of the contribution is cited.

2. Work done in collaboration with others where it is difficult to distinguish who has contributed with: it will be explicitly stated in the text that the work reported is done together with other people.
3. Work done by project students or M. Sc. students supervised by the author: the particular project report or M. Sc. report is cited.

4. Own contributions where nobody else have been involved.

Acknowledgements

This is a thesis submitted to the Norwegian Institute of Technology for the doctoral degree ‘doktor ingeniør’. The work has been conducted at the Information Systems Group, Faculty of Electrical Engineering and Computer Science, The Norwegian Institute of Technology, The University of Trondheim, Norway, under the supervision of professor Arne Sølvberg.

In the course of this thesis several people have influenced my work. I would like to express my thanks to the following people:

- My advisor professor Arne Sølvberg for inviting me to do my doctoral work in his group and for his continuous encouragement, guidance and constructive criticism to this work. He has been a source of great inspiration and interesting ideas. Finally and by no means least he has always found the time when I needed it in spite of him being very busy.

- The members of the TEMPORA project for providing a stimulating work environment. A great deal of my work was done in connection with the TEMPORA project and the its importance for this work cannot be overestimated. In particular, I would like to thank Peter McBrien for interesting discussions on ideas and also valuable feedback on drafts of this thesis. The collaboration with him and another TEMPORA member Benkt Wangler has resulted in several joint papers.

- The members of the Information Systems Group at NTH who have been invaluable discussion partners. In particular, I would like to mention my colleagues Rudolf Andersen, Gunnar Brataas, John Krogstie, Odd Ivar Lindland, Guttorm Sindre, Vidar Vetland, Geir Willumsen and Mingwei Yang for their helpful comments.

- The students I have supervised, particularly Ingrid Kvistad and Ingunn Østevik who did a literature study on how contemporary modelling approaches deal with complexity reduction and also tested an intermediate version of the complexity reduction approach suggested in the thesis.

Trondheim, December 1994

Anne Helga Seltvet
Chapter 1

Introduction

This chapter sets the context for the work reported in this thesis. It defines basic concepts and motivates the use of complexity reduction in information systems development.

Sections 1.1 to 1.3 define basic concepts in information systems development such as information system, tame and wicked problems, model, and abstraction. Section 1.4 presents the idea of complexity reduction in information systems modelling. Section 1.5 describes three case studies which will be used throughout the thesis. Section 1.6 illustrates the use of complexity reduction in the modelling process.

1.1 Information Systems Engineering

Information Systems Engineering is the discipline of developing and maintaining information systems. An information system can be defined as [148]:

The totality of all formal and informal data representation and processing activity within an organization, including the associated communication, both internally and with the outside world.

Thus, the development of an information system is closely connected to the organisations it aims to serve. The rapidly changing business environment now found nationally and internationally and the hard competition to survive, have made the information systems crucial to the success of most companies. The right information, in the right form, and at the right time is crucial to staying in business. The increasing complexity and evolving nature of organisations mean that systems development approaches must cope with a highly complex and changing reality. The deficiencies of Information Systems Engineering approaches to deal with this reality are reflected in the well known ‘Software Crisis’. Although substantial progress has
been made with respects to technology, systems development still suffers from low productivity, high development and maintenance costs and delays in delivering on time, e.g., [43, 60, 103, 44].

1.2 Tame and Wicked Problems

Information systems development spans everything from implementation of well-known routines such as inventory systems and payroll systems to large-scale heterogeneous systems, e.g., world-wide stock exchange systems. To distinguish between different types of problems, H. Rittel classifies problems as tame and wicked [106]. Tame problems can be exhaustively formulated, can be distinguished from their solution, can be tested, have a natural form, and their solution lends itself to reuse. Implementation of well-known procedures belong to the class of tame problems. In contrast to tame problems, wicked problems have the following characteristics [106, 124]:

- A wicked problem has no definitive solution.
- Every formulation of the wicked problem corresponds to a statement of the solution and vice versa.
- A wicked problem can always be solved in a better way.
- A wicked problem can be considered a symptom of another problem.
- A wicked problem is essentially unique.

Both organisational development and large-scale systems development belong to the class of wicked problems. In this thesis, only the latter is addressed.

1.3 Taming Problems Requires Understanding

'The Trick Lies in the Taming'

In [124], A. Solvberg and D. C. Kung state that the trick in solving problems lies in the taming. Basically, taming means to acquire sufficient knowledge to solve the problem. The necessary knowledge to solve tame problems is available and typically, the traditional systems approach is used to solve such problems. For wicked problems, we cannot expect a cure to be found, but rather a set of means that are tailored to support the development of wicked systems. Contemporary approaches provide a wide range of support to tame the 'wickedness' of information systems development, each emphasising different problem areas or development phases. Some areas of particular relevance are:
CASE tools [56, 107]: the method and tool aspects are emphasised.

- Requirements Engineering/Conceptual modelling approaches [104, 124]: the modelling aspect is emphasised.

- Business Process Reengineering approaches [51]: organisational aspects are emphasised.

The trend has been to automate or support more development activities and address the problems at earlier phases. The later a mistake or omission is discovered the more severe are its effects. Thus, much effort is being put into developing more comprehensive CASE tools covering organisational, modelling, method, and tool aspects. Both the PPP and the TEMPORA projects belong to such efforts.

The Concepts of Abstraction and Model

A key point in providing support for development of wicked systems is to tame the problem as much as possible by gaining insight about the problem and possible solutions. Thus, to deal with the 'wickedness' requires that the actors (e.g., developers and end-users) involved in the development process, must have sufficient understanding of those parts of the world that may have an impact on the target information system as well as on its environment. Two issues arise:

- How is this understanding achieved?

- How can the actors' understanding of the world be reflected in a computer system?

A hypothesis is that people understand the world by building mental models, e.g., [125]. These models are simplifications or abstractions of the real world. Whether the human brain actually deals with information in terms of mental models is not important for our work. The important issues are that it is impossible for the human brain to perceive all the details of the world, and that the concepts of model and abstraction have been shown to be useful in order to gain understanding about the world. The actors involved in the development process use models as a basis for communication to gain insight into the problem domain. The models are elaborated until they contain sufficient details to provide the basis for coding the software.

Abstraction In Information Systems Engineering, the concept of abstraction is crucial. It constitutes the major complexity reducing technique ever known in computing. In the widest sense, abstraction is underlying all modelling. We cannot put the reality as such into the computer, it must be simplified — we only model the major features. Moreover, we cannot consider all details at one time but concentrate
on subsets that includes the relevant details that are necessary to carry out a task. In choosing relevant subsets, we apply abstraction more or less consciously.

In the literature, the use of the concept differs. Often ‘abstraction’ is used synonymously with information hiding which was introduced as a concept in the early seventies [98].

In dictionaries, ‘abstraction’ is defined as follows:

- The principle of ignoring those aspects of a subject that are not relevant to the current purpose in order to concentrate more fully on those that are [95].
- The act of separating or removing [57].

Based on this we will define abstraction as follows:

The process of separating relevant and irrelevant details from a context.

‘Separating relevant and irrelevant details’ means that irrelevant details are suppressed whereas the relevant details are highlighted. The ‘context’ may be parts of the real world or, one or more models. Thus, in the systems development process, abstraction is used as a principle or technique to focus on those issues that are of relevance for performing a particular task and to represent these in an adequate way. ‘To focus on issues’ may involve that some specificational details are removed or separated.

Model and Specification The term ‘model’ is also widely used in Information Systems Engineering and it is used to denote a number of different things. Typically, it is used to refer to a representation of a system (i.e., a product of the development process) as well as to the language for system representation. In this work we only refer to the former as a model (and specification) and the latter is called a modelling language. A language consists of collection of constructs and a grammar\(^{1}\) (syntactical part) and an interpretation (interpretation) which define the meaning of constructs and combinations of primitive constructs into higher-level constructs.

A conceptual model can be defined as [124]:

An abstract representation of the real world phenomena that are of interest to the application in question.

\(^{1}\)The grammar (rules) shows how to form higher-level constructs from lower-level constructs.
1.4. The Idea of Complexity Reduction

A model may be represented in a human brain or represented explicitly in a computer, on a piece of paper, etc. In this thesis, we will refer to the documentation of a model as a specification. Thus, during the development process the model of the system may evolve and the model evolution is documented in a set of specifications.

The Dilemma of the Duality of Specifications

Specifications of information systems play dual roles in the systems development process: A specification serves as a basis for communication and understanding and, for coding of software. This duality of specifications is illustrated in Figure 1.1.

![Figure 1.1: Duality of systems specifications.](image)

Specifications must contain enough details and they must be understandable for the actors in the development process. For tame problems, this duality does not cause any severe problems because existing modelling approaches provide sufficient mechanisms to deal with the limited amount of detail included in such specifications. This is in contrast to large and complex systems where the amount of detail is overwhelming and it is impossible for one person to have a complete understanding of the entire system. Hence, the dilemma of the duality of specifications contributes substantially to the wickedness of information systems development. As we will see in Chapters 2 and 3, very little research is done with respect to how to fulfill the dual roles of specifications in the modelling process.

1.4 The Idea of Complexity Reduction

In the previous section, we stated that the concepts of abstraction and model (specification) are central for dealing with the ‘wickedness’ of information systems development. The increasing complexity and evolving nature of organisations and thus, information development will not reduce this wickedness, rather the opposite. The trend is towards larger systems and therefore, it is essential that systems development environments address how the dual role of specifications can be balanced under such conditions. We want to:

*Allow all relevant details to be included in a specification and at the same time make it comprehensible for the actors that participate in the*
modelling process.

A major challenge is to overcome the problems of fragmentation and complexity explosion. This has motivated our work in seeking new ways of reducing complexity in specifications by elaborating on the idea of abstraction. In the sequel, we give a more precise definition of the concept of complexity reduction and explain why we have chosen to approach the problem of balancing the dual role of a specification in this manner. In Section 1.6, we illustrate the use of complexity reduction in the modelling process.

**Specification Complexity**

Before we elaborate on the idea of complexity reduction, we need to define what we mean by specification complexity. According to Webster’s II Dictionary, ‘complex’ can be defined as [57]:

1. Consisting of composite parts.
2. Intricate.
3. A whole composed of interconnected or intricate parts.

Thus, to say that something is complex means that it consists of composite parts that may be interconnected in some way. If the number of parts and interconnections are low, a person usually perceives a specification as having low complexity (Figure 1.2a). As the number of parts and interconnections increases, the complexity of the specification also increases (Figures 1.2b and 1.2c). At some point the number of parts and interconnection reaches a level where it becomes difficult to grasp the meaning of the specification\(^2\), e.g., Figure 1.2c.

We can thus conclude that specification complexity is closely related to the number of parts and the number of interconnections between these parts. Other factors\(^3\) such as how the parts and their interconnections are represented will also affect the complexity of the specification. Thus, specification complexity is not only due to the wicked nature of such systems but also due to features of the modelling approaches\(^4\). We classify specification complexity according to what causes it:

- **Domain complexity.** This is due to the complexity of the problem domain, i.e., target domain. The domain complexity is influenced by factors such as number of parts and interconnections, the humans’ understanding of the parts and their interconnections, etc.

---

\(^2\)We may talk about parts and interconnections of a specification expressed using a non-diagrammatic language as well as a diagrammatic language.

\(^3\)This is further elaborated in Chapters 2 to 4.

\(^4\)F. P. Brooks [18] refers to this as the essentials and accidents of systems development.
1.4. The Idea of Complexity Reduction

![Diagrams showing complexity reduction]

Figure 1.2: Complexity of specification is dependent upon the number of parts and interconnections.

- *Language complexity.* This is due to the complexity of the specification language. The language complexity is influenced by factors such as how well suited the language is for modelling the problem domain, the user-friendliness of the language, the representation of the language constructs, etc.

How specification complexity can be measured (i.e., by means of metrics, e.g., [92]) is not a topic of this thesis. The complexity issues that are focused are identified based on experience using the PPP approach and TEMPORA approach for modelling of large systems from different domains (see Chapter 3).

### Complexity Reduction in Modelling

Complexity reduction can be defined as:

A process of producing *comprehensible* views of a specification in such a way that they include the necessary details in a suitable form and in such a way that they can be used *actively* during the development process.

Here 'details' refer to parts and interconnections contained in a specification. A 'view' means an abstraction of a specification. Thus, the concept of abstraction forms the basis of the complexity reduction approach. 'Comprehensible' means that the view should be appropriate as a basis for communication and understanding among the actors involved in the systems development process. By using views 'actively' during the development, we mean that they should not only serve as passive components or projections of a specification or set of specifications (e.g., for presentation purposes) but also serve as basis for *entering* new information into specifications. An *approach* based on complexity reduction provides support for dealing with complexity in specifications along with an appropriate modelling process. Thus, such an approach comprises a set of languages, methods, and tools. Figure 1.3 illustrates the idea of complexity reduction in information systems modelling.
As we will see in Chapter 4, complexity reduction demands facilities for ensuring an adequate representation and presentation of the information at the appropriate level of detail at any time of development. Representation deals with the modelling languages used to describe the system and emphasises expressiveness and formality. Presentation deals with how the system specifications are used as a means of communication among the actors and emphasises expressiveness and user-friendliness.

**Why We Introduce Complexity Reduction**

There are two major reasons for introducing complexity reducing features [115]:

1. To cope with the complexity of the Universe of Discourse.
2. To cope with the complexity of a particular specification language.
1.4. The Idea of Complexity Reduction

To cope with domain complexity  The use of models and specifications are essential in dealing with the massive amount of details involved in large-scale systems development. To model a large and complex problem domain implies large and complex specifications. As pointed out in [71], *a specification paradigm cannot change an application's complexity, it can only move its description from one level to another.* Irrespective of modelling language, the number of specificational details will increase with the size and the complexity of a system.

During the development process we must at some point specify all relevant details of the system. The complexity of the system will decide where and when the details are to be considered. As more details are gathered and represented by appropriate modelling formalisms, we may distinguish between the following cases which all call for complexity reduction:

- *Space constraints of the medium.* The specifications may grow too large to be shown at the same time, e.g., at a screen of a work station.

- *Different interests and roles of actors.* The specifications may contain details which may be of little interest to all of the involved actors. Different actors may also need different presentations to be able to understand a specification due to, e.g., different backgrounds, skills, education levels and interests.

- *Conflicting views of reality.* Different actors may hold slightly different views of the reality and the system to be built. Thus, different parts of the specifications may be based on slightly different and inconsistent views of the reality. To resolve potential conflicts among these views, it may be useful to model these as intermediate results. In some way, the final information system must accomodate a compromise of all these views.

To cope with language complexity  Specification languages may also contribute to increase specification complexity. Features such as high formality, high expressive power, and lack of structuring mechanisms may decrease the comprehensibility of a specification. Furthermore, bad choices concerning constructs of the language may also cause problems. The number of details may become overwhelming and a messy specification may result.

Formal languages are often difficult to comprehend because the constructs of the language are too abstract. In some situations it may therefore be desirable to suppress certain complexity increasing constructs. How the features of a specification language may impact on specification complexity, is further elaborated in Chapters 2 to 4.
1.5 Case Studies

In this section, the examples that are used throughout the thesis are briefly presented. Relevant details of the examples are described in Appendix ?. In the thesis, the examples are used as basis for identifying and illustrating problems of large-scale information systems development, for evaluating the PPP and TEMPORA approaches, and for exploring new ideas.

The approach of this thesis is not aimed at a particular domain. The examples are consequently taken from different domains:

- Post Office domain: the Sweden Post case study
- Library domain: the Library case study
- Oil Exploration domain: the Oil Processing case study

The Sweden Post Case Study

The Sweden Post case study [100, 133, 136] was being developed as a collaborative effort within the TEMPORA project. The Post is a large government held company responsible for the transportation and delivery of mail nationwide in Sweden. The subject of the case study was the order and invoicing system for the mail delivering part of the Post.

The author was involved in the following parts of the case study:

- Capture including initial meetings with the representatives from the Post.
- Modelling, design, and implementation using the TEMPORA CASE tool.

The order and invoicing system aims at supporting the Post’s business with large customers, i.e., customers that send large quantities of mail. The major functions of the system are shown in Figure 1.4.

The Post offers a wide range of articles (e.g., postal fees for 10 g letters) and each article has a normal price. However, the normal price can be overruled by agreements. An agreement is customer specific. It specifies what discounts the customer should be given and the conditions that must be fulfilled by the customer, e.g., the minimum quantity of mail and how it should be paid.

Each mail delivery from a customer is received together with a number of delivery notes and one covering letter. A delivery note specifies the details about the articles
in the delivery. The covering letter contains general information about the delivery such as number of delivery notes, number of delivery note lines for the various delivery notes, and serial number of the registration post office.

To get paid for its articles, the Post produces a claim to the customer based on the information in the delivery notes and in the agreements. A claim gives rise to an invoice which is sent to the customer for payment. One invoice may specify articles from one or more mail deliveries.

Based on the information provided from delivery notes and the Post’s internal accounts, e.g., data about invoices and payments, various types of statistic are produced. For instance, sales statistics are requested by sales and marketing departments at different Post’ regions.
The Library Case Study

The library case study was done by K. Weidemann Løvseth based on the design documentation of an operational library system [75]. The problem domain is the library information system at the University of Trondheim (called BIBSYS).

The author of this thesis was not involved in the initial development of the case study, but has used Løvseth’s material as a basis for testing out complexity reduction approaches.

A library information system aims at supporting all the major activities at the library. Major activities in a library are (Figure 1.5):

- **Acquisition**.
- **Administration**.
- **Public services**.

Acquisition consists of ordering, assessment, and cataloging. A document (e.g., book or report) is considered for purchase if it is suggested by a client or a librarian, e.g., after review of brochures from librarians or by shortage of copies of popular documents. Suggestions for acquiring documents are compiled in acquisition lists. The lists are checked to see if the documents exist and they are also compared to the library budget. Decisions are made whether a document should be ordered for inspection, for purchase, or not be ordered at all. Finally, orders are sent to the library’s regular vendors according to its rules for purchase.

The library receives ordered documents and non-ordered documents, e.g., gifts and documents from publishers. The documents are inspected upon arrival. Documents may be returned, e.g., documents that are damaged or assessed to be of no value, they may be cataloged, or reviewed for purchase by university employees.

Administration includes a number of activities such as registration of clients, decision about acquisition of documents, and management of loaned documents.

The public services include mainly general information services and assisting clients that want to borrow or return documents. Anyone whether registered or not, can use the general information services, e.g., accessing the library databases from public terminals. Documents may be copied, regulated by the Norwegian law.

The Oil Processing Case Study

The oil processing case study was done by M. Sandøy [111]. The problem domain is the design of a processing unit of a platform for oil production in the North Sea.
Figure 1.5: DFD specification of the library system.

The author of this thesis was not involved in the initial development of the case study, but has used Sandøy’s material as a basis for testing out complexity reduction approaches.

The oil processing unit of an offshore platform receives oil and gas together with water from the well and separates the different components from one another. The major objective of the design activity is to arrive at a device that produces as much oil as possible within the frames given by weight and spatial constraints of the platform.

The design of the oil processing unit is based on a design basis, and on experiences which at later stages also include simulation results. The design basis comprises requirements from the government, data about the wellhead, volume specifications, product specifications, information about reservoir, and the engineering company’s own requirements to process design. The major functions of the design of an oil processing unit (Figure 1.6):

- Decide number of separation steps and compressor steps.
- Estimate needs for cooling and heating.
- Estimate needs for pumps and scrubbers.
- Determine operational conditions for all the equipment.

Figure 1.6: DFD specification of the oil processing system.

The processing equipment consists mostly of separators (vertical separators are called scrubbers), compressors, heat exchanger, gas dehydrators, and pumps. Separators are used to separate the oil phase from the gas phase whereas compressors are used to increase the pressure of the gas flows which are carried in various pipes. Heat
exchangers are used to change the temperature of the process flow. Gas dehydrators are used to dehydrate gas. Pumps are used as a mean to increase the pressure of the process flow. The design of the oil process should combine these components and give them operational conditions (i.e., pressure and temperature) in a way that the final design process meets the product and volume specification given by the oil company.

The designer concentrates on pressure and temperature parameters and based on these, proper equipment is chosen to achieve the desirable pressure and temperature values. The actual design of the processing is determined by the process type (e.g., oil process) and the compositions (compounds) of the well stream. The latter varies from field to field as well as during a field’s life time. In each case, there is a need for an expert to examine each case separately to make decisions about what processing design that is appropriate. The design (e.g., separator conditions) may also be revised as a result of simulations of the proposed design.

1.6 Examples of Application of Complexity Reduction

This section illustrates the use of complexity reduction in the modelling process using the library example. It shows how complexity reduction can be applied to support three important processes in conceptual modelling:

- The *elicitation* process.
- The *conceptualisation* process.
- The *validation* process.

To illustrate the use of complexity reduction in the above processes, we will use the following scenario: Assume that we have developed a data flow diagram (DFD) of the acquisition activity of the library example, as shown in Figure 1.7. This diagram is expressed using regular DFD constructs [42]. DFDs are generally considered to be user-friendly. It has nevertheless a number of flaws such as lack of expressive power [19]. For instance, in Figure 1.7 there is no distinction between triggering and non-triggering flow. Nor is it shown how the different flows relate to each other. Below we will show one way of adding this information and validate the specification using complexity reduction. The details about the approach are described later in the thesis.
Figure 1.7: DFD specification of the acquisition activity.

The Elicitation Process

Elicitation is the process of extracting information from various sources and representing it in an informal language. Often natural language or a simple user-friendly modelling language is used. The information may be had from people as well as from available documentation. Whether documentation is available or not, a comprehensive interaction between the system developers and system buyer/system owner ought to take place. The quality of the interaction between the actors involved is very much dependent upon to which extent the communication aspect is supported.

Elicitation is also referred to as knowledge acquisition, information gathering, or information capture.
during the development process. The actors involved in the elicitation process may be representatives from different managerial levels, may have different backgrounds, skill levels, etc. Thus, elicitation support must be flexible allowing information to be viewed from different perspectives.

The elicitation process also needs to be supported at all levels of detail. In the early stages of development the information can be characterised as high-level and vague. This information is gradually refined and in the later stages, the information is low-level and operational.

**Example using complexity reduction** To add information about triggering flow and relationships between flows, only parts of the diagram are relevant at a time. A triggering flow is depicted by a thick arrow and relationships between flows are depicted by port symbols. For instance, to add ports to process P4.6 Arrival Inspection, only P4.6 and its immediate components are important. To use the full diagram in this case is confusing. Figure 1.8 shows a simplification of Figure 1.7 where only the relevant information is shown. This simplification is used in the interaction process with the librarian to elicit the necessary information. The information about control flow and how the flows relate to each other may be expressed using natural language, e.g., informal rules. An example of such a rule:

A document is inspected upon arrival. If the document is broken or assessed to be of no value for the library, it should be returned.

**Figure 1.8:** Process P4.6 and its immediate components.
The Conceptualisation Process

Conceptualisation means encoding the information formally. It includes formulating and structuring the information resulting from the elicitation process. To conceptualise relevant parts of the real world is to represent the domain knowledge using constructs of a conceptual language. This is mostly done by the developers since most modelling languages are too difficult to be used directly by the end-users. Often it is not obvious how information should be expressed in a formal language. The developers need to discuss different alternatives, why particular constructs are chosen, etc.

Large-scale information systems development implies that the developers must cover a large domain, where one cannot expect that there will be one person who has a complete overview. Thus, the conceptualisation process requires that the communication between developers is supported. This is in contrast to elicitation and validation, where the interaction process to a large extent takes place as an interaction between developers and users.

Example using complexity reduction  The simplification shown in Figure 1.8 together with the informal description of processing rules are used as basis for the conceptualisation process. In this process, the information about control flow and how the flows relate to each other are encoded formally by adding some constructs to the DFD called triggers and ports\(^6\), respectively. However, adding this information to the diagram in Figure 1.7 will clutter the diagram substantially and we cannot include all these details to it at one time. Rather than including the information in the full diagram, the developer can add the new details to the simplification in Figure 1.8. The result of this process is shown in Figure 1.9. This information may then be propagated to the full specification (Figure 1.7). If these steps are repeated for all the processes in the diagram, the result may be as shown in Figure 1.10.

The Validation Process

In the process of validation, we ensure that the specification actual reflects the users needs and requirements [20]. Arriving at 'good' specifications can be viewed as a process of negotiations between the developers and the users. To improve the validation process is very much to improve communication and understanding among the actors. Accordingly, a crucial factor here is the level of support provided for the interaction process between the people involved. The system developer needs to understand the application domain very well and the system specifications need to be presented to the users in a suitable form.

\(^6\)The concepts of triggers and ports are described in detail in Chapter 3.
Example using complexity reduction  All the ports and triggering flows are included in Figure 1.10. By using these constructs, we have added considerably more details compared to the diagram shown in Figure 1.7. The diagram appears rather messy, and it is difficult to get an overview of the acquisition activity using the diagram as it is shown in Figure 1.10. The diagram is large and complex and thus, difficult to understand. The specification's usefulness as a basis for communication and understanding among the actors involved in the validation process is reduced.

To validate the diagram using complexity reduction, we generate various simplifications of the diagram, e.g., Figures 1.11 and 1.12. These can then be used in the interaction process with the users. When the different parts are understood then it may be easier to understand the full diagram. A typical way of working will be to go back and forth between the simplified views and the full specification in an iterative manner.
Figure 1.10: Specification of the acquisition activity including triggering flows and ports.
Figure 1.11: A view of the acquisition activity where only control flow is shown.
Figure 1.12: A component view of P4.6 with the details of ports abstracted away.
Chapter 2

State-of-the-Art Survey

This chapter gives a survey on how state-of-the-art modelling approaches deal with complexity.

Section 2.1 suggests a classification of complexity reducing facilities. This is used as a basis for identifying complexity reducing facilities in contemporary modelling approaches. Section 2.2 concentrates on major properties of modelling languages and associated methods, whereas Section 2.3 provides an overview of complexity reducing properties that are implemented in commercial ICASE\(^1\) tools, i.e., tool properties. Section 2.4 shows how complexity reducing facilities relate to other modelling features of ICASE tools as well as how ICASE tools relate to contemporary CASE technology. We have chosen to focus on ICASE tools because both PPP and TEMPORA belong to this class of CASE tools.

2.1 Classification of Complexity Reducing Facilities

This section suggests a classification of complexity reducing facilities. It presents major complexity reducing techniques and describes how such techniques can be realised in a development environment. The section also outlines how complexity reduction is related to other comprehensibility increasing facilities found in conceptual modelling, i.e., validation support.

\(^{1}\)Integrated Computer Aided Software Engineering (ICASE).
Complexity Reducing Techniques

In Chapter 1, we defined abstraction as the process of separating relevant and irrelevant details from a context. It is a principle or technique to focus on those issues that are of relevance for performing a particular task and to represent these in an adequate way. Thus, to support abstraction means to allow information to be expressed at increasing levels of details. An example of a well-known abstraction is generalisation/specialisation hierarchies found in most data-oriented modelling languages. Such constructs allow a developer to include major features of an entity (e.g., name and sex of a person) as well as more detailed features (e.g., profession and details about profession). The former is found 'higher up' in the hierarchy whereas the latter are found further down in the hierarchy.

Another example of an abstraction is the use of black-box descriptions. A system is described with a set of black boxes for specifying interface details of components and hiding the details of their internals. As the developers successively gain more knowledge about the system, the internals of the components are added to the specification, e.g., a black-box is substituted by a set of new black-boxes to describe more aspects of the system. At the lowest abstraction level the internal of a black-box is typically described in detail in a precise language, e.g., using a formal modelling language.

Abstraction is a general and superior technique and the definition does not say how details are highlighted or suppressed. We have studied a number of modelling approaches and we divide techniques to reduce complexity of specifications into the following types:

- *Information hiding*
- *Structuring*
- *Viewing*
- *Translation*
- *Layout modification*

In certain cases it is rather difficult to determine which group a specific complexity reducing technique belong to. However, most complexity reducing facilities exhibit specific features (principles) which makes it easy and useful to distinguish between different groups of techniques.

**Information hiding** was introduced as a concept by D. Parnas in the early seventies [98]. It is a criterion for dividing a design specification of a software system into modules. A module, i.e., data and procedures, is designed in such a way that it
2.1. Classification of Complexity Reducing Facilities

hides its decisions\(^2\) from other modules. A module’s interface or definition is chosen to reveal as little as possible about its inner workings [98]. Thus, information hiding is a way of achieving *independence* among software components.

Information hiding can be used in a similar manner for reducing complexity in conceptual models. Information can be *encapsulated* to limit its access from other components, i.e., to create independent components. Components that need the information can access it whereas it is *hidden* for other components. In addition, the internals of the component is hidden for all components, only the interface details are available. By encapsulating information that is likely to be changed, it can be changed without affecting other components that interact with it. Complexity reduction is thus achieved by preventing components to be disturbed by irrelevant information. A component is made comprehensible in its own right without requiring insight into other components.

An example of information hiding is the use of abstract data types [49]. An abstract data type consists of data and operations upon the data. From other components, the structure of the data is not visible and the data is accessed through the operations. Another example is objects in object-oriented languages.

**Structuring** In Webster’s II dictionary, a structure is defined as [57]:

1. Something made up of a number of parts assembled in a particular pattern.
2. The way in which parts are combined or arranged to form a whole.
3. Something constructed, as a building or bridge.

In our context the ‘whole’ or the ‘something constructed’ is a specification. The ‘parts’ of the specification are expressed using constructs of the modelling language. What ‘pattern’ the assembled parts of the specification form, is dependent on the criteria upon which related parts are connected. For instance, diagrams that result from structuring information around processes in process-oriented languages will be substantially different from diagrams that result from structuring the same information around objects in object-oriented languages.

Structuring contributes to complexity reduction in two ways. We distinguish between *connectivity* and *grouping*. Connectivity addresses how components are connected in specifications to avoid isolated components as well as unconnected components which are related. Grouping addresses how related components can be placed together to achieve modularity in specifications, that is, it contributes to split specifications into more manageable and comprehensible units. From this we can define structuring as:

\(^2\)This holds particularly for decisions that are likely to be changed.
The process of connecting and/or grouping related information according to some explicitly defined criteria.

It includes techniques for connecting (connectivity) as well as grouping (modularity) related components of a specification at the same level of abstraction and across such levels. The means to structure a set of specifications will be referred to as structuring mechanisms.

**Viewing** Viewing aims at increasing comprehensibility of a specification by illuminating different aspects of it. It encompasses techniques for viewing specifications from multiple perspectives. We mentioned above that one cannot consider all details to be included in a specification but concentrate on subsets that include the relevant details that are necessary to carry out a task. Viewing techniques provide means to generate appropriate subsets of specifications.

We distinguish between persistent and non-persistent viewing. Persistent viewing means that the result of applying the viewing technique is a simplified view\(^3\) which has its own representation. Thus, a persistent view results. Non-persistent viewing means different techniques for looking at a specification without creating any persistent views. Such viewing techniques are quite common in advanced graphical user interface packages, e.g., pan, scrolling and zooming.

**Translation** A specification expressed in one language is translated into another language which is more suitable for the task to be undertaken. The rationale is that certain representations are more comprehensible and/or easier to manipulate than others. A translation preserves the semantics of the initial specification. We distinguish between translations that can be performed automatically and those that need manual interventions.

Visualisation is an important way of increasing the comprehensibility of a specification. For instance, a collection of textual rules is much more difficult to comprehend than their visual counterpart. Thus, a complexity reducing translation is to convert parts of rules into their visual counterpart\(^4\). Another example of a complexity reducing translation is to paraphrase or generate explanations in natural language for parts of a specification [48, 46].

**Layout modification** The purpose of layout modification is to produce a simpler and more comprehensible arrangement of the details of a specification without

---

\(^3\)Such a simplification of a specification will later on be referred to as a viewspec in the complexity reducing approach proposed in this thesis. In the literature, other terms are also used to refer to a simplified view such as views [29] and viewpoints [70].

\(^4\)All details are however, not suitable for showing in a diagrammatic form. This is elaborated in Chapter 4.
changing the semantics. This is in contrast to translation which implies that a specification expressed using one language is converted into a specification expressed using a different language. The layout of a specification is changed by redrawing the components of a diagrammatic specification or by rewriting the components of a non-diagrammatic specification.

To provide computerised support for producing 'nice looking' layout of diagrams is difficult. Although new graphical packages\(^5\) provide much support in drawing diagrams, little automated support is available for semantics preserving rearrangements of the details in such specifications. However, much research has focused on development of automatic layout algorithms which are used to produce 'nice' drawing of graphs on a screen without user intervention, e.g., [27, 128, 15, 88].

### Complexity Reducing Facilities

The techniques described above may be embedded in a development environment in several ways. The means to realise the techniques are referred to as complexity reducing facilities. These may appear as main concepts of the language (e.g., decomposition in DFD), extensions of the language to deal with knowledge chunks expressed using the language (e.g., structuring mechanisms), guidelines for how to carry out the development, versioning of specifications, built-in facilities in tools, etc. Complexity reducing facilities can be realised in three ways:

- **Properties of conceptual modelling languages.** We can view the languages as a vehicle for expressing all kinds of details. How these details are represented, is essential for using specifications expressed using these languages as a basis for communication and understanding.

- **Properties of methods.** A method includes support of the process of applying a language to develop systems.

- **Properties of tools.** Tools are provided to automate or support the languages used and associated methods.

The classification is not necessarily complete and orthogonal but it provides us with means to discuss the variety of features of a development environment that may have some impact on complexity. It will be used below for the presentation of how contemporary approaches to information systems development deal with complexity.

---

\(^5\)There are different categories of graphical editors from those that support independent graphical objects (e.g., idraw) to more advanced editors that support object connections (e.g., StP) and automatic graph drawing (e.g., Verilog). The latter type is mostly available in special purpose graphical packages.
Complexity Reduction versus Validation

Validation should ensure that the specification really reflects the user needs and his/her intended statements about the system [20]. A crucial factor in validation is the level of support provided for the interaction process between system developer and user. The system developer needs to understand the application domain very well and the system specifications are to be presented to the user in a suitable form. Often, the user does not know what she wants and alternative solutions, consequences of the different alternatives, etc. have to be carried out. A common understanding of the specifications is therefore a prerequisite for successful validation. Thus, most complexity reducing techniques can also be considered as validation techniques by the fact that they increase the comprehensibility of specifications by reducing their complexity. As we will see later in this thesis and also stated in [154, 73], complexity reducing techniques can be used separately as a validation technique itself or together with other validation techniques, e.g., explanation generation [46, 154, 73].

An overview how the techniques are related and address the duality of specifications, is shown in Figure 2.1. In the same way as validation techniques consist of more than complexity reducing techniques, complexity reducing techniques consist of more than techniques that are useful for validation, e.g., viewing which are useful for conceptualisation.

2.2 Conceptual Modelling Approaches and Complexity

This section provides a survey of complexity reducing facilities in state-of-the-art modelling approaches. The approaches include those that are still being researched and some that have been applied in mature systems development environments. In this section, we concentrate on major properties of languages and associated methods. The next section provides an overview of complexity reducing properties that are implemented in commercial ICASE tools, i.e., tool properties.

For most modelling approaches we concentrate on the complexity reducing features of the languages. Only major features of methods are outlined. The reason is that it is practically impossible within the scope of this text to evaluate exhaustively how associated method support\(^6\) contributes to deal with complexity to the same level of detail. It would have required thorough experiments using the approaches on real applications. Both the lack of availability of tools and the big effort to learn how to use them, prevent us from doing such experimentation.

Modelling approaches can be divided into classes according to what language per-

\(^6\)The same is true for the tool support outlined in the next section.
Figure 2.1: Overview of complexity reducing techniques and how they relate to validation techniques.
spectives (orientations\textsuperscript{7}) they support. The approaches chosen for presentation are those which are contain visual and/or rule based languages. To give a broad picture of what features the state-of-the-art approaches accommodated, we have chosen to focus on the following approaches:

- Data-oriented approaches (ER [23], ERC+ [126]).
- Process-oriented approaches (DFD [42], CAPS [78, 77]).
- Object-oriented approaches (OOA [24], Wirfs-Brock [155], OMT [110], OSA [32]).
- Rule based approaches (XCON [7], Sapiens [112], COMEX [86, 149], Viewpoint Resolution [70]).
- Transition based approaches (BNM [124], Statecharts [52, 53]).
- Hybrid approaches (ERAE [28], Gist [64], ARIES [63], Abstraction-based software development [12]).
- Other approaches (ViewPoint-Oriented Development [38], HICONS [117], knowledge acquisition approaches, flowcharts [124]).

The reason for focusing on visual and rule based languages is that these form the basis for the PPP and TEMPORA languages. Thus, the survey may provide useful input for improving the PPP and TEMPORA languages' abilities to deal with complexity. We do not describe all the modelling approaches to the same level of detail. Some modelling approaches provide more extensive complexity reducing features and these are emphasised.

Tables 2.1 and 2.2 summarise the complexity reducing features of the modelling languages described above. The tables will also be used as a basis for evaluation in Chapter 10. Complexity reducing features which are not supported by any of the approaches are left out in the tables (layout modification is not included in none of the tables and translation is included only in the second table).

Some facilities can be classified into more than one class and they are therefore repeated in the table.

**Data-Oriented Approaches**

Data-oriented languages are used to describe static aspects of UoD. They have their origin in the ER (Entity-Relationship) developed by Peter Chen [23]. The basic concepts of the ER are: entity, relationship, attribute and value. The statements 'A department has employees' and 'Both departments and employees have names', are modelled in Figure 2.2a.

\textsuperscript{7}The perspective/orientation is decided by the core concepts and principles of the language.
<table>
<thead>
<tr>
<th>Approach</th>
<th>CPFS</th>
<th>DFD</th>
<th>EER+</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>View consistency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hierarchical consistency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>泡沫视图</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Navigational structures</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary views</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Association</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Composition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relationship</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Viewing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structuring</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method Support</td>
<td>TLE</td>
<td>PRO</td>
<td>KA Approaches</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----</td>
<td>-----</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td><strong>Processing</strong></td>
<td>Association</td>
<td>Concentration</td>
<td>Development</td>
<td></td>
</tr>
<tr>
<td><strong>Zoom-in/Zoom-out</strong></td>
<td>Classification</td>
<td>Reflection</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Corresponding</strong></td>
<td>Association</td>
<td>Concentration</td>
<td>Development</td>
<td></td>
</tr>
<tr>
<td><strong>Viewpoints</strong></td>
<td>Association</td>
<td>Concentration</td>
<td>Development</td>
<td></td>
</tr>
<tr>
<td><strong>Modules</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>ADTs</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Views</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Histories</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Multiple Perspectives</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td>** større det**</td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Features</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Hierarchies</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Sections</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Refractive</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Decomposition</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Complex</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Object Class</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Object hierarchy</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
<tr>
<td><strong>Object Tree</strong></td>
<td>Development</td>
<td>Environment</td>
<td>Environment</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.2**: Complexity reducing features in modelling approaches (cont.)
2.2. Conceptual Modelling Approaches and Complexity

![Diagram of data-oriented languages: a) Basic ER, b) Generalisation in ERC+, c) Aggregation in ERC+]

The ER has later been extended with a variety of features to improve the expressiveness as well as its ability to model larger systems. Such extensions are commonly referred to as ‘semantic data models’. Examples are: ERC+ [126], ERT [83], NIAM [89, 147], PhM [122, 123, 93, 157] and SHM+ [17]. Reviews of data-oriented languages is found in [99, 58].

In general, data-oriented languages provide limited means to split up a model into more manageable units and specifications expressed using a data-oriented language are flat. Modelling a large system using a data-oriented language implies that the diagram becomes large and thus, difficult to read and understand. What most data-oriented languages do provide, is a set of general hierarchical data modelling abstractions which allow information to be expressed at increasing level of detail. The same constructs can also be considered as means to structure related components within a diagram. The most common hierarchical abstractions are [145, 99, 16, 117]:

- **Generalisation**: The differences among similar entities are ignored to form a higher order type in which the similarities are emphasised. Generalisation is often represented by an is_a relationship but other alternatives also exist. For example, the statement ‘A salesman is an employee, as is a secretary and a manager’ is modelled in Figure 2.2b by the diagrammatic representation of ERC+ [126]. ERC+ supports two types of generalisation, the is_a relationship and the may_be_a relationship. These correspond to the classical generalisation [145] and classical generalisation excluding the inclusion dependency...
between a subtype and type, respectively.

- **Aggregation.** Relationships between low-level types are grouped into a higher level type. Thus, it provides a means for specifying attributes of a new entity type. Aggregation is often represented by an is_part_of relationship. For example, an employee has an identifier and a name, one or more family names, may have a CV and keep one or more positions. In Figure 2.2c, this is modelled using the concept of complex object in ERC+ [126]. Also aggregation of entities into complex entities exists in languages such as ERT [83].

- **Association.** A relationship between member entities is considered a higher level type. Thus, it is a mechanism for defining a new entity type whose value will be a set of entities of a particular type. Association is often represented by an is_member_of relationship. For example, the set of male secretaries is an association of secretary entities.

- **Classification.** A collection of entities, i.e., instances, are considered a higher level entity type. Classification is often represented by an is_instance_of relationship. For example, the highest_paid_employee entity type consists of all employee entities which have salaries greater than 250000 N.kr. An employee which has a salary = 400000 N.kr. is an instance of highest_paid_employee.

Overviews of hierarchical constructs in static modelling languages are found in [99, 117].

As already illustrated above, most data-oriented languages provide a visual representation of the core concepts. The representation differs in different approaches. The most common examples are generalisation/specialisation and aggregation graphs, as shown in Figure 2.2. With respect to complexity, the nature of such graphs are of particular relevance: hierarchies versus networks and cyclic versus acyclic graphs. Hierarchical representations are easier to comprehend than a network. The same holds for acyclic graphs compared to cyclic graphs. For example, the basic ER [23] supports only networks of relationships. Large specifications expressed using such a language are far more difficult to read and understand than specifications expressed using more advanced languages, which provide an explicit expression of hierarchical relationships. Such hierarchies together with modern graphical user interfaces, ease the user-friendliness of specifications. Examples of functionality, e.g., navigation support in static models, provided by graphical user interfaces in ICASE tools are outlined in Section 2.3.

Some data-oriented approaches also provide mechanisms to allow definition of different views of a specification. These mechanisms can be used in a similar manner as how SQL views [25] are used to define various views of an SQL application. For example, the PrM provides the concept of scenario\(^8\) [121] for defining different views of a domain (described in the next chapter).

\(^{8}\)Scenarios have recently been renamed and is now called conceptual views.
Views developed by different developers or developed at different points in time may be overlapping or even worse conflicting. For some approaches, view integration techniques are provided, e.g., equivalence algebra for the ERC+ [96]. A survey of current developments of view integration is found in [40].

Process Based Approaches

Process based languages take as a starting point the description of the processes of the system [124]. The core concepts in process based languages are: processes and flows. Their basic form is usually a DFD. They describe the operation of an information system as a set of cooperating processes that interact via flows. DFDs were developed in the 1970s by several researchers more or less independently. Among the most well known are DeMarco [26] and Gane and Sarson [42]. As a result, several slightly different conventions have evolved. To mend the deficiencies of the regular DFD such as low expressive power and ambiguity, several extensions of the DFD are developed [77, 83, 47].

In process based languages, decomposition is the most powerful abstraction mechanism. Such languages model the interactions between processes at the same level of decomposition and the way processes at any level of decomposition relate to their parent process as well as their decompositions (Figure 3.8). Processes are decomposed until an appropriate level of decomposition is achieved. Thus, decomposition enables large specifications to be split into more manageable units according to functionality. In spite of being the subject of much debate, e.g., [19], it has proven to be useful in practice and a number of methods are based on functional decomposition, e.g., [42, 135]. An overview of hierarchical constructs in dynamic modelling languages is found in [117].

CAPS The CAPS approach [78, 77] is based on a language called PSDL. It allows information to be represented at different abstraction levels as well as mechanisms to relate information across levels. PSDL is based on DFD augmented with multiple views, where a distinction is made between facilities for providing overview pictures (summary view) and facilities for focusing on certain details (navigation structures and focused slices) [77]. A summary view is shown in Figure 2.3 (adapted from [77]). It serves as an introduction to some aspect of the system under development and to establish a context for further explanation or detailed examination. Navigation structures are used to determine more details about a particular aspect of a summary view. They include both exploding views and annotation views. An exploding view of a component shows the structure of its immediate subcomponents. In Figure 2.4a (adapted from [77]) operator A is decomposed. An annotation view gives symbolic or textual information about the selected component (Figure 2.4b, adapted from [77]).

Focused slices are subsets of one or more views formed to highlight specific information or relationships. Examples are slices that show timing, exception, critical paths and rooted sources and sinks. The two graphics in Figure 2.5 (adapted from [77])
show a highlighted critical path slice and a schedule congestion graph. The latter illustrates the intervals between the earliest and the latest time an operator can start executing.

To maintain consistency in a multi-level, multi-view system, CAPS deals with two types of consistency: hierarchical consistency and view consistency. **Hierarchical consistency** checks are necessary since adding or deleting objects requires continued top-down modifications to lower-level views. Any modifications to the graphic representation will require regeneration of the PSDL link statements along with other associated slice information to maintain the **view consistency**.
Object-Oriented Approaches

During the last decade, object-orientation has been one of the hottest buzzwords. A lot of confusion exists in the area and many claim that they have an object-oriented approach. Often an approach is referred to as object-oriented if it supports [24]: *classes* and *objects*, *inheritance* and communication with *messages*.

The focus of object-oriented languages is actors\(^8\) or objects which perform various actions towards or as parts of the information system. An information system is viewed as a network of autonomous objects communicating by interchanging messages. The interfaces of the objects are defined by abstract data types. Examples of object-oriented languages are: SHM+ [17], Wirfs-Brock approach [155], OOA [24], OMT [110] and OSA [32]. [14] presents an overview of object-oriented languages and systems.

In recent years, object-oriented languages have evolved from being pure programming languages such as Simula to more advanced conceptual modelling languages. Several languages are developed which contain a diversity of extensions. As we will see below, little is done with respect to providing overviews of how the message exchange takes place in large systems, where a large number of objects interact. Thus, it is rather difficult to get an overview of the actual behaviour of a system. Many problems can also be traced back to polymorphism and inheritance of properties in class hierarchies. Polymorphism means that the same message can be used in different ways according to the method in the class that responds to it.

Three concepts are central with respect to how object-oriented languages manage complexity: *encapsulation*, *information-hiding* and *generalisation*. Encapsulation means that both data and the operations that affect the data are grouped into a single object, as depicted in Figure 2.6a. Encapsulation constitutes the basic aggregation facility of object-oriented languages.

![Diagram of encapsulation and information-hiding](image)

**Figure 2.6:** a) Encapsulation and b) Information-hiding.

Information-hiding means that an object has a public interface and a private representation, as illustrated in Figure 2.6b. The private sides of the objects can be changed without affecting the rest of the system. The coupling between objects can therefore be restricted to the public interface.

\(^8\)Object-oriented languages are also called actor based languages.
Generalisation is provided through object class hierarchies, that is, an object class can be divided into object subclasses, where the subclass inherits the properties (i.e., data structures and methods) of the superclass.

To give a more detailed picture of what complexity reducing features the state-of-the-art object-oriented approaches accommodate, we have chosen to focus on OOA [24], the approach developed by Wirfs-Brock et al. [155], OMT [110] and OSA [32]. As we will see below, the two former has developed additional facilities to enhance the comprehensibility of specification but are rather weak in modelling behaviour. The two latter approaches are more powerful languages and focus also on modelling of behaviour.

**OOA** In OOA by Coad and Yourdon [24], the basic aggregation facility has been complemented with the concept of *whole-part structure*. The whole-part structure makes it possible to specify what objects (*class* or *class-&-object*) that make up a more complex object (*class* or *class-&-object*).

To increase the comprehensibility of large specification, the notion of a *subject* is introduced. A subject is a mechanism for controlling how much information of a model a reader is able to consider and comprehend at one time [24]. Subjects are used to look at narrow parts as well as to give overview of diagrams in an OOA model. It may be considered as a vague composition facility which allows grouping of related components. Various abstraction levels are provided by defining the notion of *layers*. A layer is made up of *class-&-object*, attribute, service, structure (whole-part and gen-spec) and subject (collapsed and extended). By showing or hiding constructs of the OOA language, different layers and thus abstraction levels result. The developer may then choose what details to see by choosing what subject to examine and then decide what constructs that should be shown, i.e., choose a particular layer. Figures 2.7 and 2.8 illustrate the use of subjects and layers.

![Figure 2.7: Subject layer.](image)

Functional decomposition is also adopted by OOA. It is used to model services associated with an object.

**Approach by Wirfs-Brock et al.** In Wirfs-Brock et al.'s approach the concepts of *contracts* and *subsystems* are introduced to group operations and objects at different levels of abstraction. The complexity of a large application is dealt with by
decomposing it into subsystems and treating those subsystems as classes. A subsystem can be decomposed into as many levels as one wishes. Figure 2.10 shows a subsystem that contains three classes. A subsystem groups related classes and subsystems into a higher level object. Subsystems are groups of classes and other subsystems that collaborate among themselves to support a set of contracts.

Contracts are used to deal with polymorphism. The responsibilities of an object class are grouped in a contract. Thus, a contract defines a set of requests that a client can make of a server. Client and server are roles that an object can play to fulfill contracts. A contract specifies what is done, not how it is done. In this way, the contract serves as a complexity reducing facility by hiding internal representation of the contract and showing only an external interface. The contracts for the Drawing subsystem is shown in the collaboration graph in Figure 2.10.
To keep track of what contracts a subsystem or a class supports, the notion of cards is introduced. Every class and subsystem have their own card. On each class card one writes the superclasses and the subclasses. In addition one lists the responsibilities that the class fulfills. If the class needs to collaborate (interact) with another class to fulfill its responsibility, one writes down the name of the collaborating class (or subsystem). This is illustrated in Figure 2.9a for the class Drawing. It has no superclasses or subclasses.

For each subsystem one writes down the contracts supported and the delegation to the internal class or subsystem that actually supports the contract, as shown in Figure 2.9b for the Drawing subsystem.

![Figure 2.9: An example using class and subsystem cards. a) Drawing class card, b) Drawing subsystem card.](image)

Subsystems are also used together with collaboration graphs to simplify the communication patterns between classes. A collaboration graph displays the collaborations between classes and subsystems in graphical form, as shown in Figure 2.10a. A collaboration is a path where information can flow. A collaboration graph is thus a description of all the paths where information can flow. To simplify the collaboration graph one can remove subclasses (subsystems) from the graph to view collaborations defined by superclasses (subsystems), as illustrated in Figure 2.10b and c.

![Figure 2.10: An example using collaboration graphs. a) The detailed graph, b) A simplified graph, c) The overall system.](image)

To get a better picture of the inheritance relationships, hierarchy graphs and venn diagrams are used. A hierarchy graph is a tool that presents a graphical representation of the inheritance relationships between related classes, as illustrated in
Figure 2.11a. A venn diagram, as depicted in Figure 2.11b, shows which responsibilities are common between classes, indicating where abstract superclasses should be created. A class can here be viewed as a set of responsibilities.

Figure 2.11: An example using a class hierarchy graph and a venn diagram. a) Class hierarchy graph for Drawing element, b) Venn diagram for Drawing element.

OMT The Object Modeling Technique (OMT) is developed by Rumbaugh et al. [110]. It is based on three languages: the object model\(^\text{10}\) the dynamic model and the functional model. The object model is used to describe static aspects of the system and corresponds to an extended ER language. The functional language is an extended DFD. The dynamic model describes the state transitions of the system being modelled.

Similarly, to the object-oriented approaches above, the object model supports encapsulation, information hiding, generalisation/specialisation and aggregation. In addition, association is supported. Moreover, structuring is supported in the object model through the concepts of module and sheet. An object model may consist of one or more modules and each module is a logical construct for grouping of classes, associations and generalisations [110]. Naming must be unique within a module but components can be referenced across modules. However, one strives for less links between the modules than within a module (information hiding is applied).

The dynamic model allows nesting of states, corresponding to state generalisation. It also provides construct for concurrent subprograms and splitting/synchronisation of control. The functional model allows decomposition/composition of data values and duplication of flows.

A system modelled by OMT is divided into subsystems in a similar manner as the approach by Wirfs-Brock et al. However, the OMT subsystems consist of a set of classes, associations, operations, events and constraints. A subsystem defines an

\(^{10}\)In OMT, the concept of a model corresponds to what we call a language. In this description, we use the OMT notation.
'identifiable' part of the system and communicates with other subsystems through a well defined interface. The lowest level subsystems correspond to modules. Subsystems are again organised as sequences of horizontal layers or vertical partitions. Each layer relays on the layers at lower levels, whereas each partition constitutes a subsystem which is providing one kind of service.

**OSA** Object-Oriented Systems Analysis (OSA) is developed by Embley et al. [32]. OSA consists of three languages: object-relationship model and object-behaviour model and interaction model. The object-relationship model can be seen as an extended ER language. It supports generalisation/specialisation, aggregation and association. The object-behaviour model is a state net. Object interaction models are used to model interactions between objects.

To deal with large and complex systems, abstraction (through the concept of **views/high-level object class**) and structuring (integration of the models) are supported. OSA distinguishes between exploded and imploded views. Figures 2.12 and 2.13 provide some examples of high-level views. Higher level abstractions are used to represent fundamental system concepts, whereas lower level abstraction unfold supporting detail [32]. The concept of high-level views can be used to structure all kinds of modelling constructs supported by OSA, e.g., object class views, relationship set views, state views and transition views. Figure 2.12 and 2.13 show\(^{11}\) two different ways of creating high-level object classes called independent and dominant object class views, respectively. Independent object class views are created by subsumption under a class name chosen independently, whereas dominant object class views are created by subsumption under a name of one of the object classes in the high level object class.

![Diagram](image)

**Figure 2.12: Creation of a high-level object-class view.**

\(^{11}\)We have used thick dashed lines for high-level views instead of shading.
Rule Based Approaches

Rule based languages represent the knowledge about the UoD by a set of rules. A rule is a law or custom which guides the behaviour of a domain. Rules may be vague or exact, simple or complex. Some rules may even contradict each other and mechanisms are then provided to resolve the conflicts. They are usually expressed in logic or a subset of natural language, e.g., stylised version of English. Such languages are often referred to as logic based (or deductive) and production rules, respectively. Emphasis is often on what the system to be modelled is going to do rather than on how to do it. Examples of rule based approaches are: MYCIN [21], XCON [7], Nexpert [87], ERL [80, 137], VWP! [70], Sapiens [112] and COMEX [86, 149].

In general, rule languages have high expressive power but they have shown to be difficult to use for large-scale development [71]. Rules tend to be complex and few approaches have provided support for dealing with large and complex rules and large and complex collections of rules. What makes pure rule based languages difficult to apply for large industrial-sized applications are their lack of [71, 7]:

- Explicit control structures.
- Abstraction mechanisms.
- Visual features.

However, several efforts have been made to mend these deficiencies and successful results have also been reported such as XCON at Digital [7] and COMEX [149]. We briefly present features of some approaches which show the wide range of means that can be used to address complexity in rule modelling.

**XCON** XCON [7] is an expert system for configuring computer systems and networks at Digital. The general structure of a rule is:
if condition then action

In addition to condition and action elements, a rule contains attributes.

XCON rules tend to be complex and complex rules need to be broken down. In particular, multiple tasks need to be factored out and each task needs to be made into an explicit, separate process. How this should be done is described in a methodology called RIME [7]. Rule complexity is reduced by splitting complex rules into simpler rules with a less number of conditions and by limiting functionality per rule. This is achieved using control techniques and subgroup schemas.

The control techniques are parts of the XCON problem-solving method and encompass situation recognition, deliberate decision and algorithmic control [7]. The two first techniques are used situations when choice must be made among competing alternatives. Situation recognition is used if the conditions and actions are straightforward and limited. Deliberate decision is most useful for managing complex situations with many variations and interactions. Algorithmic control is encouraged if the task to be implemented can be accomplished through a sequence of steps with little variation. The developer makes decisions at development time about what should happen in what order and when during execution. Although the control techniques are mainly developed for finding the rules to apply for execution they also provide some organisation of the rules.

The control techniques do not provide sufficient structuring properties and there may still be hundreds of rules within a step in a method for which the sequence of their activation is irrelevant. XCON requires that the criteria for grouping should be explicitly identified and recorded in subgroup schemas. These schemas provide abstractions which are describing sets of rules that allow developers to index into the rule base. Actions are distributed among many rules and these rules are grouped according to their functionality.

SAPIENS  SAPIENS [112] takes a different approach to deal with rules. Business rules are attached to data definitions and stored as objects. A specification expressed using SAPIENS consists of isolated objects. Thus, information hiding and encapsulation are the most important complexity reducing facilities. The objects can be specified in any order and they are highly reusable. SAPIENS uses a message-passing architecture to communicate between objects. The rules are triggered whenever a message arrives to the object. To limit the number of rules, SAPIENS applies a technique called ‘Positive thinking’. The rules are defined only from the positive point of view to avoid specifying explicitly all the negative consequences which otherwise have to be specified.

COMEX  The Control Model Editor and Execution Tool (COMEX) is a tool used for editing and executing task models [86, 149]. COMEX is a tool in the Knowledge Engineering Workbenck (KEW) which was developed in the ACKnowledge
2.2. Conceptual Modelling Approaches and Complexity

project [4]. The task model is based on the Process Model (PrM\textsuperscript{12}) which is an extended DFD language. The concept of task corresponds to a process in DFD. Each task is associated with a set of rules and an intermediate version of the coupling\textsuperscript{13} between the process based and rule based approaches in TEMPORA was used a basis for providing an overall structure of rules. In a similar manner as the coupling between the process based and rule based approaches can be transformed into a set of temporal rules in TEMPORA, COMEX uses the same mechanisms but to model knowledge based systems [149].

**Viewpoint Resolution** [70] by Leite et al. proposes a way of dealing with conflicting rules in the modelling process in order to validate rules. Rules are expressed using VWPl [70] and a set of rules constitute a view. Mechanisms are provided to compare two different views of a given situation in order to identify, classify and evaluate discrepancies between the viewpoints and integrate the solutions into a single representation, as illustrated in Figure 2.14 (adapted from [70]). A strategy for viewpoint analysis is shown in Figure 2.15.

![Figure 2.14: Viewpoint Resolution.](image)

**Transition Based Approaches**

The core concepts in transition based languages are: states, events, and state transitions. They are most commonly used for specifying real time systems. Examples of transition based languages are: Petri net [101], Behaviour Net Model [124] and Statecharts [52, 53].

**BNM** BNM (Behaviour Net Model) [124] is an extension of classical Petri nets and supports both structural and behavioural modelling. By applying the principle of *constructivity*\textsuperscript{14}, a BNM can be abstracted from a lower level model to a higher

\textsuperscript{12}PrM has also formed the basis for the PID in TEMPORA and is detailed in the next chapter.

\textsuperscript{13}This coupling is described in Chapter 5.

\textsuperscript{14}Constructivity was introduced as a fundamental principle for systems work by B. Langefors [69]. It suggests that systems work should be divided into four separate tasks: definition of a system
Figure 2.15: The strategy for viewpoint analysis.

level model. Figure 2.16a and b (adapted from [124]) The figure shows a BNM for a payroll processing system. In the lower level model, Figure 2.16a, we have a network where all of the transitions have their pre and postconditions. The algorithm for abstraction [124] finds all the transition sequences, eliminate internal variables and produces a model at a higher level (Figure 2.16b), with one precondition and one postcondition for the whole network.

Statecharts Traditional state transition diagrams have the same shortcoming as traditional Petri nets in that they are flat. Decomposition is not supported and for large systems the number of states is massive. Thus, large state transition diagrams are difficult to read and understand. Statecharts [52, 53] is an extension of traditional state transition diagrams and provides mechanisms to deal with large specifications.

Abstraction facilities are provided through two hierarchical abstractions: generalisation and aggregation. Generalisation is shown in Figure 2.17a where D is a generalisation of A and C. This means that if you are in state D you are in either A or C. Aggregation is shown in Figure 2.17b where Y is aggregation of A and D. This as a set of parts, definition of the system structure, definition of the system components and determination of the properties of the system.
Figure 2.16: BNM for a payroll processing system. a) Lower level BNM, b) Higher level BNM.

means that if you are in state Y you are in both A and D.

Figure 2.17: Hierarchy (XOR) in Statecharts. a) shows an XOR-free equivalent of b).

Figure 2.18: Orthogonality (AND) in Statecharts. a) shows the AND-free equivalent of b).

Three other features are provided in Statecharts to deal with complexity:

- **Zooming.** Zooming-in and zooming-out can be illustrated using a simple example taken from [52, 53]. Zooming-in is achieved by looking 'inside' state D in
ments. In graphical declarations, it appears as a box surrounding part of a declaration, as shown in Figure 2.24 (adapted from [37]), which defines the context C1 and C2.

Contexts provide information hiding by limiting the visibility of names. Statements within a context may refer only to names visible within this context, except for value types, which are visible everywhere. Within context C1 in Figure 2.24, the only visible names are A, R, B, f, plus all value types, i.e., XXX and YY. Within context C2, the visible names are D, g, XXX and YY. Contexts are able to communicate through sharing of events (EV) or entities (EN).

Contexts can also be organised in hierarchies, one context containing other smaller contexts. Statements can be embedded in contexts, so then they do not reside in one large and flat rule base.

Gist Gist is an operational language. It includes features from several different computational paradigms: object-oriented, process-oriented, logic-based, constraint-based and rule-based paradigms [64].

A Gist specification is comprised of two separate specifications. One specification describes the external behaviour of the whole system and the other describes the decomposition of the system. It specifies the activities of each component, the information that belongs to each component and the restrictions of interaction between the components.

There are different definitions of Gist’s semantics. In M. Feathers definition [33], the
basic concepts are states, transitions, objects and relations. A Gist specification is defined as a set of histories which denote the allowed behaviour of a system and its environment over time. Each history is composed of an initial state and a sequence of transitions, as illustrated in Figure 2.25. A state is modelled by means of objects and relationships between objects.

![Figure 2.25: A Gist specification.](image)

In a Gist specification, the states can be shown just as an ellipsis hiding the details or with the object and relationships between object (Figure 2.25). A Gist specification can also be translated into a more comprehensible representation. The Gist Paraphraser [64, 129] translates Gist specifications into natural language, i.e., English. It makes specifications understandable for non-Gist experts and assists in validation by presenting a different point of view.

Gist provides the specification with a certain degree of freedom by allowing the description of a system in a declarative manner and by offering a wide range of mappings from a high-level specification to a low-level specification.

ARIES The ARIES [63] environment allows multiple views of its knowledge base. It distinguishes between representation and presentation. The ARIES approach uses a single internal representation for all information to be modelled and allow different presentations for viewing aspects of information. The internal representation is expressed in a semantic network like language. The presentations allow information to be expressed in different notations, e.g., Gist, ERSLA, and state transition diagrams. The presentations can also be used to enter information into the knowledge base as well as to modify directly existing information. Thus, an update of a presentation can result in changes to other presentations.

The process of constructing a presentation is divided into three independent activities [63]: extraction of a subset of relevant objects from the knowledge base to be presented, layout of the overall display of those objects to be presented (e.g., directed graphs, presentation lists, matrix and pages) and portrayal of the individual objects within that overall display (e.g., shape and size of object and textual descriptions of contents). Each object to be presented consists of an icon display and a content
display. Translation is used to input information into the knowledge base and to generate the textual contents of a presentation.

The knowledge base is structured by means of workspaces and folders. A workspace denotes the context in which a developer is working on a particular problem. Each workspace consists of a set of folders and each folder consists of a set of declarations, i.e., type, relation, event, instance and invariant declarations [63]. The different types of declarations constitute the basic components of the representation language.

**Abstraction-based software development**  Abstraction-based software development is developed by Berzins et al. [12]. To our knowledge, this is the only approach that is entirely abstraction based. The key idea being to provide mechanisms called *black box descriptions* at different levels of abstraction which enable description of external behaviour of any system (and sub-systems) to be distinguished from the internal mechanisms that eventually are used to realise that behaviour. Thus, a complex system is described by a set of independent abstractions (black boxes) that are described, understood and analysed independently of the details that are used to implement the system. Black-box descriptions are used at different abstraction levels.

For each module, flowcharts and pseudocode are used to define the algorithms and data structures in accordance to the black-box description. In the initial approach [12], the pseudocode is extended with concepts similar to abstraction types well known from programming languages and specification languages for information systems [12]: abstract data types, iterators, state-machines, and transformers. More recent work addresses the problems of information overload in rapid prototyping of large-scale real-time systems, cf., CAPS above [77].

**Other Relevant Approaches**

The following approaches do not belong to any of the other orientations but are included because of their focus on specific complexity reducing techniques: ViewPoint-Oriented Development [38], HICONS [117], knowledge acquisition approaches and flowcharts [5, 124].

**ViewPoint-Oriented Development**  Finkelstein et al. suggest an approach to multi-perspective software development called ViewPoint-Oriented Development [38, 39, 90]. It provides an organisational framework developed to encompass multiple viewpoints (called ViewPoints) and various kinds of support to develop systems based on such viewpoints. The work is not tailored to any particular modelling language.

ViewPoints are defined as loosely coupled, locally managed, distributable objects.
that encapsulate representation knowledge, development process knowledge and specification knowledge about a system (e.g., design) and its domain (e.g., analysis) [90]. More specifically, a ViewPoint is internally divided into [38]: a style (definition of representation language), work plan (development process description), specification (the representation of the ViewPoint expressed in the style slot), domain and work record (development status).

Figure 2.26 (adapted from [90]) shows the main activities of the ViewPoint-Oriented Development method. In a similar manner as a system consists of a set of ViewPoints, a method consists of a set of method fragments. Each fragment describes how to develop a specific Viewpoint using a specific representation language and can be used to develop ViewPoints for different domains. Thus, such fragments can be reused (referred to as templates) and several ViewPoints can be instantiated from the same template, e.g., $VP_{T3}$.

![Figure 2.26: ViewPoint-Oriented Development.](image)

To integrate different method templates, inter-ViewPoint relations or rules are suggested to define the relationships between different methods' constituent templates. Such rules also form the basis for supporting consistency checking of ViewPoints [36, 91, 30].

**HICONS** HICONS [117] is a general diagrammatic framework for hierarchical modelling. It suggests how hierarchical constructs can be used in a uniform manner in diagrammatic languages. It advocates a diagrammatic representation of hierarchical constructs based on strict onion notation. It also allows the tree notation where this is found more advantageous but this is not emphasised. HICONS differs
from the onion notation used in Statecharts [52, 53] by supporting a strict onion notation (i.e., non-overlapping nodes and ability to refer uniquely to a node) and also by providing classification, association and vague decomposition in addition to generalisation and aggregation found in Statecharts. To adhere to a strict onion style and a the same time distinguish between the different hierarchical constructs, HICONS uses shape difference, i.e., different hierarchical constructs will have different shapes. Figure 2.27a shows the Hiconion for parts of the IFIP example and Figure 2.27b shows a slightly zoomed-out Hiconion for the same system (adapted from [117]).

![Diagram](image)

Figure 2.27: a) Hiconion for the IFIP example and b) shows a slightly zoomed-out Hiconion for the same system.

**Knowledge acquisition approaches** In contrast to most Information Engineering approaches, knowledge based approaches particularly knowledge acquisition approaches, emphasise how the development process can be actively supported. Work on knowledge acquisition has become more relevant recently, when we can observe a
shift in focus, from the initial transfer view\textsuperscript{16} (e.g., basic knowledge acquisition techniques [54]) to the modelling view\textsuperscript{17} (e.g., more comprehensive knowledge acquisition approaches [127, 151] and tool environments [4]).

To manage the complexity when developing large knowledge based systems, the trend has been to provide various methods for breaking knowledge into comprehensible pieces. The major knowledge acquisition approaches can be classified into four groups according to what perspective they take: mixture of function-oriented and perspective-oriented approaches (e.g., CommonKADS [151]), perspective-oriented approaches (e.g., COMMET [127]), task-oriented approaches (e.g., Generic Tasks [22]) and method driven approaches (e.g., Role Limiting Method [85]). The way of dividing knowledge into classes differs in different approaches, e.g., CommonKADS divides knowledge into strategic, task, inference and domain specific knowledge.

Knowledge acquisition approaches address both method and domain specific knowledge. They actively use the knowledge about the methods they suggest (e.g., to assist the developer in the creative parts of the modelling process) and also exploit domain specific knowledge to structure related concepts (e.g., by building ontologies of domain terms).

Flowcharts provide roughly the same expressive power as DFD. They are however, rather low level and used for modelling in the implementation realm. We will limit ourselves to a brief description of relevant work done on how to deal with complexity in flowchart specifications. Using flowcharts for modelling a software system, specification fragmentation is a severe problem because of the large number of unrelated flowcharts, each representing a specific view of the model. Auglænd [5, 124] suggested a technique for relating various flowchart specifications to each other by defining a simple formal model representing all the basic properties that are commonly depicted in flowcharts. From this model relevant views that accomodate specific features could be derived. For example, the system diagram in Figure 2.28a (adapted from [124]) gives an overall view of a system by presenting all objects and all primary relations between them. This diagram can be used as a basis for deriving views which focus on particular aspects of the system, as shown in Figures 2.28b and 2.28c, respectively (adapted from [124]).

\textsuperscript{16}In the transfer view, knowledge acquisition was considered as the problem of eliciting knowledge from an expert and represent this knowledge using an appropriate knowledge representation language.

\textsuperscript{17}The transfer view has now been modified and the knowledge acquisition problem is now considered a much more complicated task. The knowledge may be compiled or hidden and the expert may become conscious about her knowledge during the knowledge acquisition process. A model of the expert's knowledge is gradually constructed. This view corresponds very much to the view taken in conceptual modelling approaches in general.
2.3 Complexity Reducing Facilities in ICASE Tool Environments

This section provides an overview of complexity reducing properties that are implemented in commercial ICASE tools. The dual role of a specification has not been explicitly addressed in such tools. This together with the fact that their conceptual basis is rather weak [6] made us concentrating our effort on the tool aspects, which are the strong dimension of contemporary ICASE. Language and method aspects are only briefly mentioned.

The survey is based on a review reported by OVUM [6], product documentation from ICASE vendors and own experiences using some tools and demonstrations of some tools.
Structuring  Most ICASE tools support connectivity between components in the different languages by name links. The definitions are stored in a dictionary/repository which usually provide two views of its information [6]: a diagram view (i.e., objects joined by lines) and a dictionary view, i.e., nested lists of texts. To what extent these two views are kept consistent differ in different tools. The same applies to how they register the association between the components themselves. The more advanced ICASE such as IEF keep a consistent view and thus, an update of one component implies that corresponding components in other diagrams are updated. E.g., if a process in a process hierarchy is included a similar process will be included in the DFD at the corresponding level of decomposition. The cross references are taken care of by the data dictionary structure.

Some tools such as IEF and ExSpect\textsuperscript{18} have built in a grouping function in their graphical editors where a collection of symbols can be chosen (by clicking the mouse) for hiding details. The result is a composite object where only its interface can be seen in the diagram.

The structuring mechanisms provided by a tool are used as a basis for navigation. Although the advanced graphical user interfaces enable various facilities for navigation, the limitation of such facilities is the conceptual basis of the languages supported by the tool.

Viewing  Most contemporary ICASE tools, e.g., The Bachman Product Set, IEF, Foundation and ADW, are based on advanced graphical user interfaces and thus, support extensively non-persistent viewing such as pan, page, and zoom.

The Bachman Product Set provides the most advanced non-persistent viewing facility which allows subviews of specifications to be generated by specifying filters. A filter is a set of criteria that define the items to be shown. The filters can only be applied on diagrams, that is, DFD and ER specifications. It is not possible to interleave information that appear in a diagram with additional information about the corresponding components in the repository. Filters can however be applied to parts of a diagram. This allows full specification of certain details and hiding others in the same diagram. Subviews are non-persistent and cannot be included in reports. Neither facilities for managing subviews are provided nor facilities for updating such subviews.

Another interesting example of non-persistent viewing is provided by the grouping facility of IEF which is described above. It can also be considered as a useful non-persistent viewing facility by the ability to contract and expand groups of components. For instance, it is possible to contract the subtypes of an entity and show just the supertypes.

However, persistent viewing is poorly supported. An example of persistent viewing is the concept of scenario in RDD (Requirements Driven Development [2]) which sup-

\textsuperscript{18}ExSpect is developed by Bakkenist Management Consultants.
ports the generation of different views of a textual specification and allows multiple views to co-exist. The limitation of this approach is the lack of formal specifications. Other tools such as Foundation and IEF allow the user to focus on subsets of data but only allows one item type to be considered at a time.

Translation This is limited to show different styles of a modelling language. For instance, the user may choose between Gane and Sarson and DeMarco style for DFD specifications.

Layout modification Again due to the modern graphical user interfaces, ICASE tools provide some support in modifying the layout of diagrams. Functions to manipulate diagrams provided by ICASE tools can be divided into three types [6]: moving, viewing and cosmetics. Moving includes functions such as groups of components can be visually selected for moving on the screen, pick components for a group discontiguously, snap to grid, etc. Viewing corresponds to the basic non-persistent viewing functions briefly mentioned above: page, pan, zoom, fit to page printing, etc. Cosmetics include scaling of component, automatic redrawing after update, free text on diagrams, etc.

Although the modern graphical user interfaces represent a shift in how modification of layout can be performed, little automated support is available for semantics preserving rearrangements of the details in such specifications. To provide computerised support for producing 'nice looking' layout of diagrams is difficult. Existing automated layout techniques found in ICASE tools are limited to simple functions such as to provide more space at the bottom of a big tree structured diagram, to provide some support of optimisation of connection paths and to allow duplication of components to limit the number of crossing lines.

Combinations of the above techniques A striking feature of the tools is their limitation in combining the different complexity reducing facilities they support. Functions are invoked separately and they have a strict limitation on how many different types of components they can operate upon.

2.4 CASE Tools

In the previous sections, we have focused on complexity reducing facilities. In this section, we show how such facilities relate to other modelling features of ICASE tools as well as how ICASE tools relate to contemporary CASE technology. Figure 2.29 shows the context of the survey and illustrates the dependencies.
From 4GLs to CASE Tools

In the late seventies, the Fourth Generation Languages (4GLs) represented a shift from pure programming languages to more comprehensive modelling environments, e.g., Cullinet and Focus. 4GLs offered higher-level languages and additional support facilities for systems development such as graphical packages and report generators. However, they did not offer any facilities for construction and modelling of application systems and development of large systems remained a problematic issue. To mend the well known ‘Software Crisis’, the concept of CASE tools was introduced in the beginning of the eighties. A CASE tool can be defined as [107]:

All software packages that automate or support one or more activities in the systems development life-cycle.

During the last decade, several new CASE tools have appeared at the market place and a large number is being prepared at different research institutions [55, 107, 56].

Classification of CASE Tools

CASE tools are divided into [45]:

- First generation CASE tools (analyst workbenches and code generators).
- Second generation CASE tools (IPSE, CASE framework, CASE shell and ICASE).
The first generation CASE tools were point tools (e.g., StP) which provided support to certain tasks within a development phase. Typically, they provided drawing facilities, limited syntax checking and some support for generation of application code\textsuperscript{19}. From the point tools in the early 80s, the CASE technology evolved into more or less integrated solutions. Among the most well known are IPSE, CASE framework, CASE shell and ICASE. The major distinction is in their strategy to achieve an integrated solution:

- **CASE frameworks.** They provide an open framework in which various point tools can be integrated. The aim is to end up with a tool environment that covers the entire development life-cycle and in which all services are accessed through a uniform user interface. Examples are ANSI/IRDS, ISO/IRDS and AD/Cycle.

- **IPSE** (Integrated Project Support Environment). They are similar to CASE frameworks but they also provide administrative support functions such as version control and multi-user support. Among the most well known are Maestro, ISEE and ADE.

- **CASE shells.** CASE shells differ from the above in that they are meta-tools. They provide a tool environment which consists of facilities to develop CASE tools tailored to any approach. TEMPORA is developed around RAMATIC, a CASE shell developed by SISU\textsuperscript{20}.

- **ICASE** (Integrated CASE). They provide an integrated set of tools which are tailored to a particular approach. Examples of commercial ICASE tools are Foundation, IEF, IEW and Bachman Product Set.

All the above CASE tools are integrated via a central database which stores specifications from different components. Other terms used for a central specification database is data dictionary, encyclopedia and repository. These concepts may also be used to distinguish between different levels of sophistication.

The PPP and TEMPORA approaches belong to ICASE and in the remainder of this section, we will therefore concentrate on tools that belong to this category.

**General Properties of ICASE**

With ICASE, we mean a tightly connected tool environment which accommodates the following major properties:

\textsuperscript{19}Often this was limited to the generation of program skeletons. To make the system operational, the programmers manually filled in the remaining code.

\textsuperscript{20}Swedish Institute for Systems Development, Stockholm.
• *Life-cycle support.* The purpose is to have one tool environment to support the
development of an information system, from its conception to its installation
and subsequently through its evolution (the maintenance phase).

• *Method support.* An ICASE is often based on a particular methodology. The
tool should provide the necessary functionality to automate or support the
modelling activities that are included in the approach taken.

• *Phase integration.* The tool should not only support the various development
phases but also provide a smooth transition from one phase to the other.
For instance, the deliverables from one phase should be available for use in
subsequent phases. The extent to which phase integration is achieved is highly
dependent on features of the underlying specification database.

Today most ICASE tools address all development phases, at least that is what the
ICASE vendors claim. However, the extent to which the tools actually provide
adequate method support during the various phases differ greatly [6]. So is true for
phase integration [6].

**Functional Features of ICASE**

To accommodate the three major properties above, an ICASE must provide a wide
range of functionality. We have partitioned the functional features as follows:
*user interaction, integrated modelling and design support, group and project management
support, repository services and run-time support.* This classification does not nec-
essarily provide a complete framework for describing functional features of ICASE.
The main purpose is to provide a more systematical treatment of the major groups
of features and thus, show how complexity reducing facilities relate to other features
of such a tool. We will briefly describe each feature below. Since our focus is on
complexity in conceptual modelling, we will only elaborate integrated modelling and
design support in the sequel.

**User interaction** The ability of users to work successfully with a tool is depend-
ent on the extent to which the use of the tool is made easy, quick and pleasant.
Accordingly, the most striking feature of most tools today is their user interface, per-
mitting the designer to work with graphical, form based and textual input/output
in a multiple window style of interaction.

**Integrated modelling and design support** Integrated modelling support in-
cludes facilities to support the process of expressing a system's properties at every

---

\[21\] A subset of these features are described in [114]. This description was based on a classi-

cification of features proposed in [20]: user interaction, verification support, validation support,

modelling/design support and development project management.
level of detail from business modelling to design level. Design support\textsuperscript{22} includes facilities to support the process of transforming a conceptual specification into a specification which is executable or a specification from which executable code can be automatically generated, i.e., code generation. Integrated modelling and design support is further elaborated below.

**Group and project management support** Group and project management support are essential for making ICASE tools work effectively in larger, decentralised projects. Examples of group and project management support are: maintenance of design decisions, communication and mail facilities, annotation management, authorisation management, version and configuration control, tracking support and support for co-working in a decentralised design environment.

**Repository services** A repository is often based on a meta-model which is an underlying definition of the repository. Thus, the meta-model specifies the structure of the repository. A repository may provide a number of services such as single user access/multi user access, security control (e.g., password), access control, version control, locking, recovery and facilities for reorganisation.

Within CASE tool development, a substantial effort is being put into the development of more sophisticated storage mechanisms. Point tools such as StP \cite{107}, have data dictionaries whereas more advanced CASE tools such as IPSEs (Integrated Project Support Environments) \cite{56} have repositories. The most ambitious concept introduced so far is AD/Cycle's repository concept \cite{56}. One of the major problems in developing such an advanced storage mechanism is to develop a comprehensive enough meta-model underlying it.

**Run-time support** Run-time support includes all kinds of support that is needed to generate a running information system from a design specification of the same system. The trend is that such a system is generated from a design specification by a code generator. However, to provide an operational system that fulfills requirements to performance, reliability, coverage, \textit{etc.}, activities such as optimisation, testing, installation of the running system in the operational environment, \textit{etc.} need to be supported. Most ICASE tools provide comprehensive run-time support, e.g., INSTALL/1.

\textsuperscript{22}The reason for making a distinction between integrated modelling support and design support is that the former is implementation \textit{independent} and the latter is implementation \textit{dependent}. However, they are described together since the support is to a great extent overlapping.
Integrated Modelling and Design Support

In addition to *complexity reducing facilities*, integrated modelling and design support conveys:

- *Analysis support*
- *Creative support*
- *Support for transformation*
- *Support for reuse*
- *Support for user interface design*
- *Support for transaction design*

We will briefly outline each of the above types of support. As we will see below, only some of these features are found in contemporary ICASE tools, the more advanced ones are still focus of research. A general description of complexity reducing facilities were provided in the previous section and will not be repeated here.

**Analysis support**  When we develop a large system we are heavily dependent on comprehensive analysis support to make sure that the specifications are consistent and formally correct. Analysis support comprises verification and validation support. The purpose of verification is to ensure that a specification is complete and formally correct. Most tools provide syntax checking but checking of semantic consistency is, however, poorly addressed.

Validation as such was briefly described in the previous section. There are numerous validation techniques developed such as paraphrasing of graphical (formal) specifications in natural language [63], explanation generation [46], generation of abstracts of (parts of) specifications [2], animation or simulation of (parts of) a specification [63], and prototyping facilities [73].

**Creative support**  ‘Creative support’ means that the tools should help the user in the creative activity of constructing a model. There are at least two kinds of creative support which are concerned with method knowledge and domain knowledge, respectively. The *method knowledge* contains information that is used to support a user in using a particular method developing a specification. It contains, for instance, advice to be given in different situations, rules for checking a specification for quality flaws, etc. The method knowledge base is for most contemporary ICASE tools, relatively small, but it is expected to grow as more experience and method knowledge is acquired. The *domain knowledge* serves to assist the specifier to use appropriate application concepts when developing a specification. This part is also
used for more advanced types of semantic and quality checking of the specification. More advanced checking can normally only be done in the context of a particular domain.

As far as creative support is concerned, most efforts have been conducted in the area of Knowledge Based Systems, cf., Section 2.2. The trend is however, that these are adopted in Information Systems Engineering approaches. Knowledge based techniques are used in current Requirements Engineering approaches such as ERAE [28] and Gist [65] and information system development approaches such as DAIDA [59] and OICSI [109].

Support for transformation This means support for transformation of specifications from one 'level' to another. Examples of transformation of specifications are: view integration, restructuring of a specification, database schema generation and transformation of a non-executable requirements specification into an executable specification. View integration is required, e.g., when combining the local specification efforts of a number of work teams working in parallel. Restructuring implies the semantics preserving rearrangement of a specification in order to improve it according to a set of quality, performance or other kinds of rules or according to a designer's restructuring directives.

Support for reuse This means support for reuse of specifications or parts of specifications. The concept of reusable specifications indicates a possible future extension of existing CASE tools. The ideas are to capitalise on components of "older" specifications so that new specifications can be designed by utilising, combining and extending a "library" of conceptual components. Components interesting for reuse may include previously developed system components and components including knowledge related to the application domain, design rationale, system architecture and hardware and software. To maintain a library of reusable components, appropriate mechanisms for representation, storage and retrieval of the reusable knowledge are necessary. Examples of such efforts are REBOOT [118].

Support for user interface design User interface design includes specification of dialogue (interaction flow) and detailed layout for input and output (screen layout). Several advanced user interface packages exist that exploit recent advances in graphical user interfaces and workstation technology. Examples are Motif, XWINDOW and TeleUse.

Support for transaction design During transaction design the transaction boundaries are identified and the transactions' internals are specified in detail. The usual database semantics of a transaction entail that it has the ACID properties [31, 146]. The support of transaction design includes facilities for specifying transaction boundaries and validating, optimising and refining transactions, e.g., type and frequency.
2.5 Chapter Summary

This chapter has identified a classification schema for complexity reducing facilities which has been used as a basis for presenting major features of contemporary modelling approaches. The survey shows that the trend is towards more comprehensive approaches which provide special complexity reducing constructs in the languages (e.g., hierarchical constructs) and provide additional facilities which are realised by method and tool support (e.g., the introduction of viewpoints). However, we could observe that none of the approaches we have examined provide support for all aspects of complexity reduction but rather they provide a limited number of features and build comprehensive support around these. Moreover, most approaches seem to focus more on facilities which deal with complexity in the resulting specifications than on the process of actually developing the specifications. Except from general guidelines, the process of developing large specifications is to a large extent left to the developers.

In the field of knowledge acquisition, support of the modelling process has been emphasised, e.g., CommonKADS [151] and COMMET [127]. Recent research in Information Systems Engineering has however begun to focus on process aspects, e.g., the viewpoint-oriented approaches such as Viewpoint Resolution [70], ViewPoint-Oriented Development [38] and ARIES [63] and more pure approaches to process modelling such as the decision-oriented approach described in [108, 105]. As far as the latter is concerned, to our knowledge little attention is paid to complexity issues and we have therefore left them out from our discussion. The viewpoint-oriented approaches address only parts of the modelling process, in particular requirements modelling, and they are still in an immature state. Thus, considerable effort is needed before they are available for commercial use.

If we compare the features of state-of-the-art languages to conceptual modelling with what are actually provided in contemporary ICASE tools, we can observe a large gap as far as complexity reducing features are concerned. The dual role of specifications is not addressed by existing tools and thus they only provide limited support. However, the requirements to more expressive languages together with the fact that the information development must deal with steadily increasing amounts of information, indicate that many of the features found in contemporary modelling languages are most likely to be found in the next generation of ICASE tools. The fact that tools that provide some support of complexity reducing features received a high score in a recent evaluation of commercial CASE tools is another indication of this direction.

From the discussions above we conclude that there is a need for approaches to information systems modelling which address complexity in the resulting specifications as well as supporting the process of developing large and complex specifications.
Chapter 3

Baseline

This chapter presents the TEMPORA approach which was developed in the ESPRIT project TEMPORA. It also describes experiences using intermediate versions of the approach in case studies. Experiences that are relevant for discussing specification fragmentation and complexity explosion are emphasised. These experiences were used as a basis for developing facilities to improve the TEMPORA approach's abilities to deal with development of large and complex information systems, which are described in Chapters 5 to 8 of this thesis.

Section 3.1 gives an overview of the TEMPORA approach and Section 3.2 provides a more detailed description of the TEMPORA languages. Experiences using intermediate versions of the approach in case studies are reported in Section 3.2.

3.1 An Overview of the TEMPORA Approach

The aim of the TEMPORA project was [131]:

"... to improve the software development process through the exploitation of an approach which explicitly recognises the role of business policy within an information system and visibly maintains this policy throughout the software development process, from requirements specifications through to an executable implementation. This implies that the TEMPORA paradigm views the development of an information system as the task of developing or augmenting a knowledge base of business rules..."

The TEMPORA languages allow system specification from three different perspectives: data-oriented, process-oriented and rule-oriented. The languages are: the Entity-Relationship-Time Language (ERT) [83, 135, 150], the PID [10, 93] and the External Rule Language (ERL) [80, 132, 135, 137]. The ERT is an extended Entity
Relationship language and describes the static aspects of the UoD. The PID and the ERL describe the dynamic aspects including the temporal dimension. The ERL is a logic based language whereas the PID is an extension of standard DFD. Rules are also used to express constraints and derivations on the static language.

In TEMPORA, information development is viewed as comprising a set of modelling levels, each having its clearly defined objectives and a number of associated means to model building and support. The levels are: Business Modelling Level, Information System Modelling Level, Design Modelling Level and Run-Time Level.

A prototype CASE tool environment is developed. It includes conceptual modelling and software engineering tools based around RAMATIC and Probe\(^1\) \[13\]. For each language, an editor is implemented in RAMATIC. Analysis tools for each language as well as their interrelationships are implemented in PROBE, where each language's meta-model is represented in the PROBE repository. The theoretical aspects of the rule-based paradigm extended with temporal aspects are exploited within a pragmatic environment through the incorporation of a commercial DBMS. The run-time platform is based around the BIM-Prolog and SYBASE RDBMS. The TEMPORA architecture including the different components and their interrelationships within a development stage and across development stages is shown in Figure 3.1.

### 3.2 The TEMPORA Languages

This section describes the TEMPORA languages: the ERT, the PID and the ERL.

#### The ERT

The major modelling constructs of the ERT are: entity classes, relationships classes and value classes. The concept of value class corresponds to an attribute in the traditional ER. The language is also extended with constructs to describe complex entity classes, generalisation/specialisation and temporal aspects. An example which illustrates the main features of the language is shown in Figure 3.2. For more detailed descriptions of the ERT, see \[83, 135, 150\].

A complex entity class is a collection of complex entities, where each entity is an aggregation of entities (complex or simple) and relationships. Thus, a complex entity C is an abstraction of an ERT sub-schema. An example of a complex entity class is given in Figure 3.2 (Operational Condition).

*Generalisation/specialisation* is supported through the *is_a* construct. The *is_a* relation has the usual set-theoretic semantics. If two subclasses belong to the same

\(^1\)PROBE is an object-oriented layer on top of PROLOG.
entity class are specialised under the same specialisation criterion, they are always disjoint.

We distinguish between total, partial and overlapping is.a relations. A total is.a relation means that all the subclasses under the specialisation criterion are included. A partial is.a relation means that some of the subclasses under the specialisation criterion are included. Total and partial is.a relations are depicted by an is.a symbol with a filled circle (Figure 3.2) and a non-filled circle, respectively. Subclasses which are not disjoint are depicted with separate is.a relations for each subclass, i.e., overlapping subclasses.

Time is explicitly modelled in the ERT by timestamping of entity classes and relationships for which the history of changes should be preserved. We distinguish between two aspects of the temporal nature of entity classes and relationships: temporal variation and historical perspective. An example of temporal variation is provided in Figure 3.2.

An entity class undergoes temporal variation if entity class instances may exist at certain ticks and not at others. This is denoted by adding a T descriptor on to the entity class box. The notion of ticks is introduced to refer to various points in time [83]. However, timestamping is not used only to indicate that some time vari-
ation occurs but also to represent the temporal variation of entities with respect to each other. This yields both relationships between entity classes as well as relations between a superclass and its subclasses. Temporal variation of a relationship means that the period which relationship involvements can exist are related to the period which the associated entities exist. Temporal variation of a is.a relation means that the period which the superclass entity can exist are related to the period which the entities of the subclasses can exist. For temporal variation, these periods must be overlapping.

In contrast to temporal variation, historical perspective can be used to relate periods of relationship involvements where the periods when the associated entities exist, are different. This is denoted with an H descriptor on the relationship.

For a mathematical definition of the temporal aspects of the ERT, the interested reader is referred to [83].

The PID

The PID is based on the process language of the PPP language\(^2\), but has been revised to fit with the TEMPORA approach. It is an extension of regular DFDs [42]. It specifies processes and their interaction in a formal way. The interaction between processes is modelled using the concept of flows. PID specifications include the inter-

\(^2\)The PPP approach is described in Chapter 9.
actions between processes at the same level of abstraction, as well as how processes at any level of abstraction relate to their parent process and their decompositions. The basic modelling concepts of a PID are: processes, external agents, triggering flows, non-triggering flows, ERT views and timers. An example of a PID specification using the constructs is shown in Figures 3.3 and 3.4. For a detailed description of the PID and PH, see [132].

Figure 3.3: Top-level PID model for oil processing design.

Figure 3.4: Decomposed PID Model for oil processing design including ports.

The concepts of process, external agent, ERT/view and data flow have the same meaning as in the traditional DFD. The external agent and the ERT/view in the PID correspond to the source/sink concept (i.e., external entity) and the store concept, respectively. To increase the expressive power the PID contains some extensions compared to DFDs.

A flow is modelled as a channel carrying items. Items are the objects stored in stores,
sent or received by external agents and carried by flows. They may be defined in
the PID and the item is the link between the ERT and the PID.

The concept of flow is extended to denote *material flow* as well as *dataflow*. No
distinction is made between flows carrying data and flows carrying material. Further-
more, flow may be *triggering* or *non-triggering*. Triggering flows determine when
a process should be considered for firing. A process is triggered when the items on
its triggering flows have arrived. Triggering of a process implies that a process *in-
stance* is created. A trigger may be triggered one or more times and each time a
new process instance is created. Triggering flows are depicted in the diagrams by
a T in the processes where the flow ends. Triggering and non-triggering items may
be *buffered* upon arrival at a process class if there is no process instance to consume
them. Such items are stored in a *buffer*.

The concept of *store* is extended to denote a *material* store as well as a *data* store.
Stores denote abstractions of repositories of information and material.

A *port* is a graphical formalism for expressing logical relations between input flows
(and output flows, respectively) of processes. A process can have one or more input
ports and one or more output ports. Ports define the set of valid combinations of
input flows and output flows which will make up a *process instance*. The ports can
be classified into three types (the graphical notation is depicted in Figure 3.5):

- **Plain ports** (AND, OR and XOR) depict that the items on the flows enter the
  process once only, i.e., in one ‘batch’. The meaning of the ports:

  - An AND port depicts a logical AND relation between the flows.
  - An OR port depicts a logical OR relation between the flows.
  - An XOR port depicts a logical XOR relation between the flows.

- **Repeating ports** (REP) depict that the items on the flows enter the process one
  or more times, i.e., in one or more ‘batches’.

- **Conditional ports** (COND) depict that the items on the flows may enter the
  process once, but only if the condition that applies is satisfied.

\[
\begin{align*}
\text{AND} & \quad \text{OR} & \quad \text{XOR} & \quad \text{COND} & \quad \text{REP}
\end{align*}
\]

**Figure 3.5:** The port symbols of the PID.

The different ports can be used isolated or nested to form composite ports. An
example of the former is shown in Figure 3.6. The process has 4 possibilities: ‘arrivals
on a’, ‘arrivals on b’, ‘departures on c’ and ‘departures on d’. The process has two
possible (valid) input combinations, either ‘arrival on a’ or ‘arrival on b’ but only one valid output possibility namely ‘arrival on c and d’.

An example of how composite ports can be used is shown in Figure 3.7. The process has 7 possibilities: ‘arrivals on a’, ‘arrivals on b’, ‘arrivals on c’, ‘arrivals on d’, ‘departures on e’, ‘departures on f’ and ‘departures on g’. There are three possible input combinations: (b) (i.e., one b flow), (a, b) (i.e., one a flow, one b flow) and (c, d) and three possible output combinations: (e, f, g), (e, g) and (f, g). This yields a total of 9 acceptable flow combinations for the process as a whole.

Timers are either clocks or delays. Clocks are used to model events that are to occur at a specific moment in time. Delays are used to model events that are delayed a certain time interval and are specified relative to the occurrence of a flow.

In addition to the PID, an overview facility called Process Hierarchy (PH) is also provided. A PH is introduced to represent hierarchical relationships between processes and their decompositions. No ordering is implied in the diagram. Each non-leaf node of a PH is described by a PID. Figure 3.8 shows the PH generated from the PIDs of Figures 3.3 and 3.4. Processes on higher levels of abstraction are pictured ‘higher up’ in the Process Hierarchy. It should be noted that each process in the PID is allowed to be represented by only one icon in the PH.

A complete PH for a given instance of a process model shows all the processes in the corresponding PIDs, i.e., all processes at all levels of decomposition. For some applications, the number of processes may grow large and it may appear useful to split
Figure 3.8: An example Process Hierarchy.

the PH into several smaller PHs, each depicting a limited part of the decomposition hierarchy.

The ERL

The ERL is a declarative rule language. It is based on first-order temporal logic and is extended with constructs for querying the ERT. Rules expressed using the ERL may both describe and constrain processes of the PID at any level of decomposition, but the language only requires them for describing the lowest level. In addition, the ERL is used to express constraints and derivations on the ERT. All ERL rules are given a single general structure:

\[
\text{when trigger if condition then action}
\]

where the when and if parts are optional. The basic elements of the ERL expressions that may appear in the trigger, condition and action fields of an ERL rule are: (1) selections of data, (2) sets of data obtained from the ERT and (3) flows which name tuples of data selected from the ERT. The when, if and then parts of an ERL rule may however, also be composite. Compound expressions are constructed by the usual connectives of classical first order logic (e.g., and) and temporal logic, e.g., sometime in past.

To give procedural semantics to an ERL rule, a rule must be categorised as being a constraint rule, a derivation rule, or an action rule. A constraint rule expresses conditions of the ERT database which must not be violated. For example, ensuring that the number of separator steps is different from 2, can be expressed as:

\[
\text{for all processing-unit (X) it follows that count \{Y for which processing-unit (X) consists-of separator (Y)\} \neq 2}
\]
A derivation rule expresses how information can be derived from information that already exists. For example, deriving that equipment is associated with a high pressure process if the equipment is heavy and expensive, can be expressed as:

\[
\text{if processing-unit (X) consists-of equipment (Y) [has weight = 'heavy', has price = 'expensive']} \\
\text{then high-pressure-process (Y)}
\]

An action rule expresses what actions to be taken if an event occurs and the conditions evaluate to true. For example, stating that upon receiving a change request provided that information about compressor conditions is available, new separator conditions will be computed, can be expressed as:

\[
\text{when change-request (Separator, Temperature)} \\
\text{if compressor-condition (Condition) and calculate-pressure (Temperature, Condition, Pressure)} \\
\text{then separator-condition (Pressure) and processing-unit consists-of separator (Separator) [has operational-conditions has pressure = Pressure]}
\]

For a detailed description of ERL, see [80, 132, 135].

### 3.3 Experiences Using the TEMPORA Approach

This section describes some of the experiences from practical usage of the TEMPORA approach that are relevant for discussing specification fragmentation and complexity explosion. The experiences are described with respect to: modelling language and method and tool support.

#### Experiences Using the TEMPORA Language

In the following, we discuss experiences using the TEMPORA language with respect to the following properties:

- Comprehensibility.
- Expressiveness.
- Complexity increasing constructs.
- Abstraction mechanisms.
Comprehensibility The basis of the TEMPORA languages is a set of well known languages: DFD, ER and logic. For those who are familiar with the basis, the TEMPORA languages as such are rather simple to learn and understand. However, the ERL is a logic based language and suffers from many of the weaknesses of standard logic\(^3\). The experiences from practical usage show that the rule language is not straightforward to use. In particular, there are some aspects that appear to cause confusion:

- **Classification of rules.** It was difficult to differentiate between the different rule types. For example, users tend to think that all the rules including a *when* part is always an event-action rule.
- **Constraints.** Some simple constraints turned out to be rather tricky to specify. Simple statements can blow up into complex forms and little guidance in rule formulation is offered, e.g., quantification.
- **Temporal constructs.** The temporal operators turned out to be rather difficult to comprehend and this caused difficulties in the modelling process.

Expressiveness The TEMPORA languages have been used in a number of case studies, e.g., [133] and these conclude that the languages have shown its feasibility with respect to expressiveness. The explicit formulation of business rules adds a new dimension of concepts and expressive power. There are however, some aspects that are not addressed such as exceptions and organisational concepts. Nor does TEMPORA distinguish between objects in the real world and information about these objects. The latter can be illustrated by an example from the Sweden Post Case Study. What the Sweden Post sells is called a product by the accounting department and an article by the marketing department. The data associated with article and product is different although both refer to the same object in the real world.

Another aspect that has caused problems is how one should deal with transient information in the PID. At an early stage it was decided in TEMPORA that all flows should be defined with respect to the ERT. As a consequence one had to add all objects appearing in flows to the ERT model, even if they were transient. In this way, objects that are sent between processes as intermediate result appeared in ERT and obviously, the ERT ended up containing objects that should not appear in such a specification.

In addition, modelling using ERL suffers from other problems which have also been reported in other studies and development projects applying rule based approaches, e.g., [71, 7, 149]:

\(^3\)Logic-based languages are characterised by being low level, computable and a standard for comparison but unstructured, unreadable and difficult to use. This is in contrast to natural language-like languages which are characterised by being expressive, flexible and easy to use but their computational ability is poor.
3.3. Experiences Using the TEMPORA Approach

- Maintenance is difficult due to lack of adequate control structures.
- Testing is difficult and costly due to the difference between control and data flow not having been modelled in rules.
- Testability and maintainability are reduced since simplification of representation of a flat rule-base is obtained at the cost of sacrificing modularity and abstraction.
- Reliability seems to be poor, judging from the difficulties found in testing rule-base systems.

Complexity increasing constructs  The added expressiveness of the PID contributes to increased complexity of large specifications. Specifying systems beyond toy-examples implies that the amount of details become large and the diagrams become difficult to read and to understand. In particular, these problems are caused by complex port structures and large number of flows, c.f., Figure 1.10.

The ERT is also a visual language, however, it is flat and two features seem to cause problems. First, there is no distinction between a relationship between two entity classes and a relationship between an entity class and a value class. Both are graphically shown and the size of the entity class and value class symbols are the same. In the Sweden Post Case Study, each entity class has several value classes. Fragmentation of specification results because of space constraints. The way of splitting up such large specifications are not supported and was therefore carried out in an ad hoc manner.

Second, large generalisation hierarchies are difficult to comprehend. Factors such as inheritance rules, time stamping and associated constraints expressed in ERL reduce the comprehensibility of the hierarchies.

ERL rules become very complex and there is a need to find ways to cope with representational complexity when using rules for modelling of large systems. Rules of the type:

\[
\text{when} \ trigger \ \text{if} \ condition \ \text{then} \ action
\]

where any of the trigger, condition and action fields may be composite, may become quite difficult to use and understand. The fields may contain flows, ERT access expressions, logical connectives, temporal connectives, or aggregates, i.e., sets or tuples. In spite of supporting user-oriented concepts, the textual form is not effective for communication and understanding when the complexity grows.
This can be illustrated with an example:

```
when covering_letter_and_delivery_notes (AgN,CuN,CuNa,B1,B2,B3,B4,S,PO,ND,NDLs)
  if post_office.P [has latest_serial_number = OldS,
    has post_office_number = PO]
  and S = OldS + 1
  and customer.C [has customer_number = CuN,
    has customer_name = CuNa,
    has office_address address
    [contain care_of = B1,
      contain street_and_number = B2,
      contain zip_code = B3,
      contain city = B4 ]]
  and T == <DN.NO,...> and T member_of NDLs
  then for all T it follows that
    (delivery_note.DN [has delivery_note_line = DN.NO,
      delivered_at post_office.P]
    and enter_dln_head (C,PO,DN,AgN,ND,NDLs))
  and post.office.P has latest_serial_number=S
  and status ("Serial number is correct")
else status ("Serial number is false")
```

The textual form of a rule makes it hard to perceive when the number of possible combinations of the different fields becomes high. There is a need to provide means to grasp the overall structure of the domain/system being modelled, something which calls for visualisation. Another fact is that many components of rules refer to components expressed in the other languages making rule modelling a tedious task. Furthermore, the experience also shows that most people working in the domain of transaction-based systems find it easier to describe the variety of tasks they carry out rather than state the rules governing the activities they are involved in [100].

The example above shows that one single rule may become hard to understand because of its size and complexity. Then we can imagine the problems of using a rule-based specification comprising a large number of rules as a basis for communication. A large number of rules which reside in a flat rule-base is difficult to manage unless comprehensive support is provided.

**Abstraction mechanisms** The ERT provides the following abstractions: generalisation/specialisation (*is_a* hierarchies) and aggregation (complex entity classes). PID provides an important abstraction mechanism by the principle of decomposition. Case studies [100], have shown that the abstraction mechanisms of ERT and PID were useful for modelling large industrial-sized applications. The ERL has no abstraction mechanisms beyond predicates and this turned out to be a serious flaw of the language.
Experiences Using the TEMPORA Method and Tool Support

The case studies using the TEMPORA languages for modelling of large-sized information systems have shown that specification complexity is a serious problem for communication and understanding among developers as well as between developers and users, e.g., [133]. These will be described as follows:

- Lack of structuring mechanisms.
- Rule relations at differing abstraction levels.
- Rigid specifications.

Lack of structuring mechanisms  The experiences from case studies show that development using the TEMPORA languages suffer from fragmented specifications. The same languages are used throughout the modelling process. However, the languages as such are poorly connected and thus, the specifications expressed using these languages are poorly connected. This fragmentation of specifications is a serious problem in the modelling process due to the fact that related (or duplicated) information may not be related in the specifications.

The languages are overlapping and partly they may be used for specifying exactly the same things. For instance, the ERL and the PID refer to objects described by the ERT and the dynamics expressed by the PID can also be expressed by the ERL. This redundancy of information increases the complexity in the modelling process and specifications become difficult to change and at the same time maintain consistency.

The same languages are also used at different levels of development. For example, the ERT describes the main concepts of the enterprise at the Business Modelling Level but the main concepts of a particular information system at the Information System Modelling Level. However, models used at different levels are not related in the current approach. The same yields models expressed using the PID and the ERL. The specification databases are simply copied from one level to the other and necessary modifications and extensions performed. No references or links between corresponding concepts at different levels are kept by the TEMPORA development environment. As a consequence, knowledge expressed at different levels of abstraction is not related.

Reviewing the results of various development projects using the TEMPORA language, revealed that the deliverables contained a high number of specifications more or less related. The various specifications were expressed using different languages. This heterogeneity in the specifications caused severe problems since the dependencies among the various languages are not well understood and no facilities are provided to deal with interrelationships between specifications.
Rule relations at differing abstraction levels. As pointed out in case studies, rule modelling was not straightforward to carry out. It was necessary to catch high-level as well as low-level rules but these were not related even if the low-level rules were refinements of the high-level ones. At an early stage it was stated that when aiming at modelling large and complex information systems there is a need for introducing means to relate rules to one another and to the other languages of TEMPORA [100]. To impose structure on a flat rule base, a set of relations were introduced [116, 156, 135]: *motivates*, *necessitates*, *refers-to*, *causes*, *overrides* and *suspends*. In Appendix A, the rule relations are briefly described together with experiences using them on examples and in a case study. We limit ourselves to summarise the major experiences:

- Rule relations were perceived as useful by the developers.
- The rule relations as defined above are partly ambiguous. Our experience is that each developer seemed to have different opinions about the meaning of the relations because their names are similar to terms often used (misused) in natural language.
- Graphical representation of rule relationships is necessary.
- The requirements to vagueness and preciseness differ during the modelling process.
- The maintenance of links can be a heavy burden.

Rigid specifications. High granularity of work modules gives rigid and inflexible specifications. In the TEMPORA approach, the granularity of a modelling object is a specification. This has serious consequences for the use of the TEMPORA languages for practical modelling. First, the languages which do not have any inherent (or have limited) structuring mechanisms such as the ERT and the ERL are difficult to use for modelling of systems beyond toy examples. If the specifications need to be split up, e.g., because of space constraints, this is often done in an *ad hoc* manner. This aggravates the fragmentation even more. Second, the languages are not sufficiently integrated so accessing relevant parts of other specifications are not straightforward to do, e.g., that may also be a large and complex specification. Thirdly, large diagrams are difficult to understand and awkward to manipulate. The problem becomes even more severe for large rules and for collections of rules.

The TEMPORA languages span two extremes: visual components (provided by the ERT and the ERL) and textual components (provided by the ERL). It does not however, provide any means to view a specification’s contents expressed using any of the languages from different perspectives beyond the languages themselves and the rule relations mentioned above. Thus, the fact that different actors may have different interests, roles, knowledge, skills, *etc.* are not taken into account. Although the visual representation in particular the PID, provided some useful scoping support, this was not felt enough. For instance, rule modelling requires deep insight
into the domain because the details are necessary to specify the rules but one is not able to deal with all these details at once.

The lack of structuring mechanisms for the TEMPORA languages implies that it is rather difficult to change a large specification and keep track of necessary changes when developing a system. Sometimes this also influences modelling decisions by preventing the developer from doing certain changes to a specification because the task would simply be too time consuming.

3.4 Chapter Summary

This chapter has presented the TEMPORA approach. Each of its languages the ERT, the ERL and the PID, have been described together with experiences using intermediate versions of the approach in case studies. Although a large number of reports have been written which describe experiences using various features of the TEMPORA approach, we have limited ourselves to present experiences which are relevant for discussing specification fragmentation and complexity explosion.

The experiences were described with respect to languages and method and tool support. As far as the languages are concerned, they were found rather simple to learn and understand for those who are familiar with the basis, i.e., DFD, ER and logic based languages. They have also shown their feasibility with respect to expressiveness although there are some aspects which are not addressed, e.g., exceptions and organisational concepts. However, experiences from using all the components together show that large specifications expressed using these languages are difficult to use and understand. Developing large information systems using the TEMPORA languages implies large and complex diagrams and large and complex collection of rules, respectively. Additional expressiveness are provided in the languages, e.g., modelling of temporal information, to enable inclusion of more details without providing facilities to deal with these details in practical usage. Ways of dealing with the massive amount of details should consequently be provided in order to deal with the complexity in information modelling following the TEMPORA approach.

The experiences reported in this chapter form the basis for identifying a set of requirements to an approach based on complexity reduction in the next chapter. Moreover, facilities to improve the TEMPORA approach’s abilities to deal with development of large and complex information systems are described in Chapters 5 to 8 of this thesis.
Chapter 4

Requirements to an Approach Based on Systematic Complexity Reduction

The previous chapter introduced the TEMPORA languages and examined their strengths and weaknesses based on the experiences using the languages on examples and in case studies. Chapter 2 provided a state-of-the-art survey of relevant aspects of contemporary specification approaches. We have now provided a basis for identifying and describing major requirements to a successful specification approach for large-scale information systems development based on complexity reduction, c.f., Chapter 1. The approach described in the next chapters will address most of them. The requirements will also form the basis for the discussions in Chapters 10 and 11 about achievements of this thesis and give directions for future work.

Section 4.1 proposes a classification of major requirements for an approach based on systematic complexity reduction. Sections 4.2 and 4.3 examine language dependent and method and tool dependent requirements, respectively.

4.1 Classification of Requirements

This section identifies a classification of major requirements for an approach based on systematic complexity reduction. To allow the specifications to contain large amounts of detail and at the same time be comprehensible, depends on:

- To what extent the language is appropriate for modelling large and complex systems.
- To what extent the process of using the language to build large and complex specifications is supported, i.e., method and tool support.
The requirements for an approach based on systematic complexity reduction must take both aspects into consideration. We concentrate on those aspects that are of particular concern when developing large and complex systems. General requirements are only briefly described, with references to other work provided.

The classification below does not necessarily provide an orthogonal or complete framework for describing requirements. The main purpose is to provide some classification of the requirements to facilitate a more systematic treatment of the major groups of requirements to an approach based on systematic complexity reduction.

Language Dependent Requirements

Many papers and research reports have been published that identify requirements to modelling languages (e.g., [58, 117, 74]) and to particular aspects of languages (e.g., [50]). To benefit from previous work, we will use the general criteria developed by G. Sindre [117] as a basis for identifying language dependent requirements. He distinguishes between two main types of requirements for language quality:

- Requirements to the conceptual basis of the language.
- Requirements to the external representation of the language.

For each of these two types, four main sub-types of requirements are identified [117]:

- *Perceptibility*: How easy is it for human beings to grasp the language?
- *Expressive power*: What is possible to express in the language?
- *Expressive economy*: How effectively can things be expressed in the language?
- *method and tool potential*: How easily does the language lend itself to proper method and tool support?

In addition, we add another requirement which are necessary for dealing with development of large systems based on complexity reduction:

- *Reducibility*: What features are provided by the language to deal with large and complex systems?

---

1By language quality we mean how good/appropriate the language is suited for the purpose of modelling information systems.
4.2. Language Dependent Requirements

Method and Tool Dependent Requirements

To deal with large and complex system specifications, we identify the following essential groups of requirements concerning method and tool support:

- **Reducibility:** How can complexity be reduced in large and complex specifications?
- **Changeability:** How easy is it to perform changes in specifications?
- **Modelling freedom:** How flexible is the approach?

4.2 Language Dependent Requirements

The requirements to the conceptual basis are similar for all languages for information systems development. In spite of being general, they provide the foundation of a successful approach based on complexity reduction.

The external representation of languages differs. We limit ourselves to dealing with visual and rule based languages. We also examine the requirements to an integrated approach based on these languages. The reason for focusing on visual and rule based languages is that these form the basis for the PPP and TEMPORA languages.

Perceptibility

**Conceptual basis** A common language is a prerequisite for effective communication and understanding among the actors involved in the modelling process. This requires concepts that are close or equal to the actors' view of reality, i.e., the conceptual basis of a language should be more or less identical or similar to the actors' own terms. The essential requirements to perceptibility are [117]: the concepts of a language should be natural and intuitive, they should be easily distinguished, the number of concepts should be reasonable and the use of concepts should be uniform and consistent.

**External representation** Requirements to perceptibility of the external representation include: the graphical symbols chosen for different concepts of a language should be intuitive, it should be easy to discriminate the various symbols, the symbol use should be uniform, the symbols should be as simple as possible and visual emphasis of symbols should be used in accordance with the relative importance of the concepts. Visual emphasis include factors such as multimedia components, letter fonts, colours, moving or blinking symbols, position of symbols and connectivity of objects.
The advantages of each language should be utilised when combining visual and rule languages to form an integrated approach. For instance, by specifying the main points using a visual language and specifying the details using a rule language, we would take the best out of each language. It is awkward to express too many details in a diagram but their abstraction facilities are often superior compared to a rule language, whereas details can easily be specified using a rule language.

Expressive Power

Conceptual basis The conceptual basis must provide a sufficient number of concepts to express what we need to but in such a way that the perceptibility requirements are satisfied. The essential requirements to expressive power are [117]: the concepts of a language must be general rather than specialised, they must be composable so more complex expressions can be formed from simpler ones and they must allow for precise knowledge as well as vague knowledge to be expressed.

External representation In general, the expressive power of the external representation should be equal to the expressive power of the conceptual basis. As far as rule languages are concerned, an explicit representation of control structures should be provided [100, 71].

Expressive Economy

Conceptual basis The conceptual basis should provide concepts which make it possible to form statements which are as brief as possible. However, the concepts must be chosen in such a way that the perceptibility requirement is met. The essential requirements to expressive economy are [117]: the concepts of a language should assure that the most frequent and important kinds of statements can be expressed as briefly as possible, whereas less frequent or less important kinds of statements may be more awkward to express.

External representation A number of factors can improve the expressive economy of a language such as defining special symbols for frequent or important symbols. When it comes to expressive economy rule languages have several disadvantages compared to visual languages, cf., ‘a picture says a thousands words’. Everything must be explicitly expressed, which leads to the textual representation being rather verbose. Thus, for an integrated approach that combines visual and rule languages, the advantages of the visual representation should be exploited if possible.
Method and Tool Potential

Conceptual basis The conceptual basis must be such that the language can be used with achievable methods and tools for information systems development\(^2\). Essential requirements to the method and tool potential are:

- **Formality.** The concepts of a language should be formally defined. The method potential increases by providing concepts which have clearly defined semantics, i.e., there is no doubt about the meaning of each concept of the language. Formality also increases the tool potential by facilitating automatic reasoning and support, e.g., automatic redrawing of graphs.

- **Efficiency.** Formality is not enough for having a high tool potential, it must also be possible to provide the tool support in an efficient way. This is particularly relevant for modelling of large systems, where e.g., automatic reasoning and redrawing of diagrams may become inefficient processes.

- **Separation of concerns.** The method and tool potential increases if the conceptual basis facilitate a natural division of the systems work into more manageable tasks, cf., the constructivity principle by B. Langefors [69]. This presumes that the reducibility requirements are sufficiently satisfied by the conceptual basis (see below). Development of large system is never a one person effort. Separation of concerns is therefore important for supporting development of system specifications by groups of people.

External representation The method and tool potential of a language is directly connected to its perceptibility. If the external representation of a language is hard to perceive, its method and tool potential is also low. For development of large systems, it is crucial that the requirements to reducibility are sufficiently satisfied. Otherwise information overload and space constraints of the medium may become serious problems.

Reducibility

Conceptual basis The conceptual basis should provide features which facilitate modelling of large and complex systems, i.e., it should provide sufficient concepts which have complexity reducing properties. The essential requirements to reducibility are that concepts of the language should:

- Allow information to be expressed at various abstraction levels, instead of forcing, e.g., a high-level business concept to be expressed in a concrete implementation-oriented representation too early in the development process.

\(^2\)With achievable we mean that associated method and tool support can be developed for the language with a reasonable effort, not presuming a major breakthrough in basic sciences or technology.
Must be composable so more complex expressions can be formed from simpler ones in such a way that the irrelevant details are suppressed. Thus, the language should provide concepts which can express superficial views as well as narrow views on details.

Express relationships between various abstraction levels. These concepts provide the glue between specifications at the same abstraction level and across abstraction levels. Such conceptual links should have clearly defined semantics rather than pure name-links. This is in contrast to contemporary CASE tools which mainly support browsing and navigation in specifications due to the limited name-links they support, i.e., syntactical basis. By providing links with clearly defined semantics, the tools can exploit the potential of formal specifications, e.g., tool kits are not only limited to support drawing of diagrams.

Group related information. The conceptual basis should not only provide concepts to links related information but also facilitate grouping of details into desirable knowledge chunks, i.e., to achieve modularity in specifications.

Enable information hiding. Thus, concepts should be provided to encapsulate information in order to limit its access from other components, i.e., to create independent components. Components that need the information can access it whereas it is hidden for other components.

**External representation** The external representation of the concepts which have complexity reducing properties should meet the same requirements to perceptibility, expressive power, expressive economy and method and tool potential as any other concepts of the language. In addition, there are some other important factors:

- Explicit representation of links between related information. Links between symbols may take a variety of forms such as explicit graphical links between related symbols in the diagrams, identification of symbols in different diagrams by unique naming and graphs showing explicit links between related symbols.

- Visualisation of links. Graphical representation is preferred to textual representation 3.3 since it is rather awkward to get an overview of the constructs that are related by the textual representation, i.e., low comprehensability.

- The links should record various relationships between information to allow for viewing the information from different perspectives.

- Redundancy may increase perceptibility.

- Ways of showing explicitly overlapping and redundant information. This is particularly important when a large diagram is split into several subdiagrams.

---

3That is, names must correspond in the specifications.
4.3 Method and Tool Dependent Requirements

An approach’s ability to deal with large and complex systems is not only dependent on the features of its modelling languages but also on what kinds of method and tool support it provides. The method and tool support should assist the developer in using the languages for building such systems by:

- Providing necessary complexity reducing facilities (which are not sufficiently addressed by the languages).
- Providing automated support where this is appropriate and possible.
- Giving guidelines/methods for how to apply complexity reduction in the modelling process.

As already described above, the quality of the method and tool support is, of course, dependent on the method and tool potential of the languages to be supported.

The requirements to the complexity reducing facilities are basically the same whether they are embedded as features of the languages or as features of the method and tool support. Requirements for complexity reducing techniques that are already dealt with under language dependent requirements are not repeated here.

Reducibility

To deal with fragmentation and information overload in specifications, requires that a sufficient set of the complexity reducing techniques are provided: information hiding, structuring, viewing, translation and layout modification. As already mentioned above, some of them may be embedded as properties of the language. This holds particularly for information hiding and structuring. However, they may also be embedded as method and tool support. For example, vague composition facilities can be built-in tools such as grouping of entities and relationships in IEF. Another example is links between different versions of a specification in versioning and configuration control systems. Although viewing, translation and layout modification are dependent on the features of the language such as formality and perceptibility, they are mostly realised as features of the method and tool support. Layout modification is often a tedious task and automated support should be provided if possible.

So far we have only mentioned viewing to a limited extent. Since it is a crucial feature of our complexity reducing approach, we will take a closer look at how it should be used to reduce complexity in large and complex specifications. It should be possible to view a specification from different (and optionally related) perspectives. The fact that the people involved cannot be considered a homogenous group but may differ in many respects such as roles in the development process, managerial level,
background and skills, advocate an approach which supports multiple perspectives. Some models will only apply to a particular part of an organisation, whilst others must be applicable all over the organisation and at the same time be related to the models of the part. Thus, we require mechanisms to support different perspectives of composite specifications and relate these to each other, i.e., relate relevant parts of the global and local models. This requires that:

- One perspective only should not be emphasised.
- Multiple viewpoints must also be allowed to coexist in specifications.
- The two extremes of viewing are on one hand overview pictures and on the other focus on a narrow set of details and both should be supported.
- Flexible generation of views and management of such views should be provided. Various views of specifications may be useful at different stages and most likely chaos would result if it is left to the developers to do it in an ad hoc manner.

**Changeability**

When developing large systems, changeability is a central issue. If no support is provided to perform changes in specifications, development of such systems is a tedious and erroneous process [79]. To allow new information to be included in a flexible manner requires that support is provided to:

- Localise place of change.
- Perform change. This also includes propagation of changes in a controlled manner if the change has effects beyond one specification.
- Manage change. To keep track of many specifications and various versions of specifications, demands for advanced versioning and configuration control facilities.

**Changeability and fragmentation** To ensure controlled propagation of change, related information must be identified and kept track, i.e., duplicated\(^4\), overlapping and complementing information. Sufficient facilities must be provided to ensure that related information is explicitly connected in the specifications\(^5\). This is crucial in the case of updates of existing information or by insertion of new information. Propagation of changes to maintain consistency in specifications would be a major effort unless related information is explicitly linked. On the other hand, allowing

---

\(^4\)Redundancy of information is a desirable feature during the modelling process particularly during business modelling [139].

\(^5\)Such facilities are later referred to as structuring mechanisms.
a number of links to be specified may substantially increase the burden on the developers. To maintain links is a major effort and in practice there is a trade off between the usefulness of the links and the effort that needs to be put in to maintain them.

**Changeability and granularity of work modules** It should be possible to work with modules at different levels of granularity. As far as visual and rule languages are concerned, this means that facilities should be provided to work actively with diagrams and rules at various levels of details. In particular, mechanisms should be provided for flexible inclusion of details in large rule bases and large diagrams. Rather than operating at the full specifications including all details, it should be possible to focus on a particular aspect of a specification without being disturbed by irrelevant details.

Manipulation of larger specifications is a tedious task. Thus, comprehensive tool support is necessary. This may include support for selecting various views of specifications as well as flexible ways of manipulation diagrams. For large diagrams expressed using languages with high expressive power, this implies that there is a need for mechanisms to deal with complexity increasing constructs, manipulation of layout, etc.

**Modelling freedom**

Method and tool support should address both the process of developing the specifications, i.e., corresponds to a *process-oriented view*, as well as describing the deliverables of the development process, i.e., corresponds to a *product-oriented view*. An inflexible approach may hamper the creativity of the developer. Thus, a modelling approach should provide a more *constraint-oriented* way of working instead of the typical prescriptive (imperative) style provided by most integrated CASE tools. Although the major goal is error-free specifications, it may, e.g., be advantageous to allow temporary specifications to be conflicting and even wrong. This is particularly useful in the initial phases of the development. As pointed out by Lindland [73], some degree of freedom is necessary for the modeller to be able to work effectively. Modelling freedom is also discussed by Feather [34].

### 4.4 Chapter Summary

This chapter has identified a set of requirements to an information systems development approach based on systematic complexity reduction. We have distinguished between language dependent requirements and method and tool dependent requirements. For the former, we divide requirements into two types: those which describe features of the conceptual basis and those which describe features of the external
representation of the languages. Both types were again divided into five groups: perceptibility, expressive power, expressive economy, method and tool potential and reducibility. Reducibility addresses explicitly complexity reducing properties of the languages and are therefore the most relevant for our purpose.

Many of the language dependent requirements are general in the sense that they would apply to any development approach whether it is based on complexity reduction or not. In spite of being general, they provide the foundation of a successful approach based on complexity reduction and hence, they were important to include to provide a more complete set of requirements to complexity reduction.

An approach's ability to deal with large and complex systems is not only dependent on the features of its modelling languages but also on what kinds of method and tool support it provides. The method and tool support should assist the developer in using the languages for building such systems by providing necessary complexity reducing facilities (which are not sufficiently addressed by the languages), by providing automated support where this is appropriate and possible, and by giving guidelines/methods for how to apply complexity reduction in the modelling process. More specifically, the method and tool dependent requirements are divided into three groups: reducibility, changeability and modelling freedom. The reducibility features may be embedded as properties of the languages and then supported by methods/tools or they may be entirely embedded as method/tool support.

To deal with fragmentation and information overload in specifications, requires that a sufficient set of the complexity reducing techniques are provided. Thus, the requirements identified in this chapter together with the features of the TEMPORA languages and the experiences using them in practice, form a basis as is how the TEMPORA approach should be extended to encompass complexity reduction, as is the topics of the next chapters.
Chapter 5

Combining the Process Based and Rule Based Approaches in TEMPORA

In the previous chapter, we identified a set of requirements to an approach to complexity reduction. Among these, requirements to facilities for providing integrated specifications were essential. Integrated specifications are not only important for reducing the problem of fragmentation but also for utilising the different languages' strengths. The experiences showed that we need to develop comprehensive support for rules if they are going to be used for modelling of large and complex systems. This chapter proposes a method of integrating process based and rule based approaches to the development of information systems. Via this coupling we can use utilise the abstraction, structuring and visual features of the PID to enhance rule modelling in the development process. As we will show in the reminder of this thesis, we consider the coupling between PID and the ERL as essential for supporting complexity reduction when the TEMPORA languages are used.

The chapter describes the coupling between the ERL and the PID both at the conceptual level (i.e., Business Modelling Level and Information Systems Modelling Level) and at the design level (i.e., Design Modelling Level). At the conceptual level the coupling is defined using the logical semantics of the languages. Based on this coupling we present a graphical representation of rules and describe how the elements of a PID can be expressed using the ERL. At the design level the coupling between the ERL and the PID is made via the executional (imperative) semantics of the ERL together with the coupling at the conceptual level.

Sections 5.1 and 5.2 define the coupling at the conceptual level and design level, respectively. Section 5.3 describes some important aspects of rule execution in a temporal database environment.
5.1 The Conceptual Level Coupling

This section details how the connection between the ERL and the PID at the conceptual level can be made by using the semantics of the PID and the ERL as a basis. The PID and the ERL were informally described in Chapter 3.

Complementary Languages

Both the ERL and the PID can be used to describe the dynamics of a system, and as described in Chapter 3, each language has its advantages and disadvantages. The motivation for the coupling between the process based and rule based approaches is to reduce the fragmentation of TEMPORA specifications. In achieving this, we will utilise each language’s strengths and use them as complementary languages for information systems modelling. Three particular areas where the strengths of one language serves to enrich the other one: Firstly, we want to have a flexible representation and presentation of rules. This leads us to have both a form based representation and a graphical representation of rules. The form based representation corresponds to the ERL. The graphical representation of rules is achieved by representing certain constructs of the rules in the PID. Secondly, we want to extend the temporal semantics of the PID to encompass the temporal aspects of the ERL. Thirdly, we want to investigate the feasibility of rules as a specification language for process logic, i.e., the internals of a process. Thus, in combining the process based and rule based approaches we exploit the user-friendliness and the structural properties of the PID and the expressive power and formality of the ERL.

Defining the Basic Coupling

The basic structure of an ERL rule is:

\[
\text{when } \text{trigger if condition then action else action}
\]

The \text{trigger}, condition, and action fields may be any ERL expression. The ERL expression can contain flows, ERT access expressions, logical connectives, temporal connectives, or aggregates, i.e., sets or tuples.

ERT access expressions are ERL constructs which select data from an ERT model (relationships are used to restrict the selection of instances from classes in some way) [82]. Flows name tuples of data (single or aggregates) selected from the ERT. They allow information to be shared between rules.

Since the ERL expressions are logical formulae that can be rewritten to DNF, thus an ERL expression \( \eta \) may be regarded as taking the form:
5.1. The Conceptual Level Coupling

\[ \eta = (\eta_{11} \text{ and } \eta_{12} \ldots) \text{ or } (\eta_{21} \text{ and } \eta_{22} \ldots) \ldots \]

The major constructs of the PID are: processes, external agents, triggering flows, non-triggering flows, ERT views, and timers. By examining the central concepts of the PID and the central concepts of the ERL, we observe that the external properties of a process of the PID may be expressed by an ERL rule. The relationship is depicted in Figure 5.1 and can be described as follows:

- The trigger part of the process structure, i.e., the triggering flow \( \alpha \) entering a process, corresponds to the when part of an ERL rule.

- The conditional part of the process structure, i.e., non-triggering flow \( \beta \) entering a process, corresponds to the if part of an ERL rule. The process logic \( \gamma \) should also be included in the if part. We mentioned above that a condition field can be rewritten to DNF. This means that if a condition \( c \) has a DNF \( c_1 \) or \( c_2 \) or ... then each of \( c_1, c_2, \ldots \) can be divided into an input part \( c_{1,\beta} \) and a processing part \( c_{1,\gamma} \), giving \( c \) the form:

\[ (c_{1,\beta} \text{ and } c_{1,\gamma}) \text{ or } (c_{2,\beta} \text{ and } c_{2,\gamma}) \text{ or } \ldots \]

The input part \( c_{1,\beta} \) may contain a flow or ERT access expression, and the processing part \( c_{1,\gamma} \) may contain ERL expressions excluding flows and ERT access expressions.

- The action part of the process structure, i.e., the output flows, corresponds to the then part of an ERL rule. The action part may contain both triggering flows \( \delta \) and non-triggering flows \( \epsilon \).

![Figure 5.1: Relationship between the PID and the ERL.](image)

We have now presented the basic connection between the ERL and the PID by describing the way in which trigger, condition, and action fields are reflected in the PID and vice versa. We now elaborate how the PID may be connected to parts of the ERL.

**Processes** A process requires information from its surroundings, that is, it interacts with other processes, ERT views, timers, and external agents via flows. A process may have one or more input flows and one or more output flows. A process' combinations of flows (input flows or output flows) are expressed using the concept of ports and control aspects are expressed using triggering flows, as described in Chapter 3. Triggering flows and ports make it possible:
- To represent permitted combinations of input/output flows for each process, i.e., the way the various elements of the trigger, condition, and action fields are logically related.
- To express information about sequencing of processes.
- To express conditional inputs.
- To express if a process always produces an output given a valid set of inputs, i.e., conditional outputs.

We will now present the way a process including ports may be described by a single rule expressed using the ERL. In the Figures 5.2 to 5.6, each construct of the diagram syntax of a PID process including ports is shown together with its logical semantics expressed in the ERL. Details on port construction in the PID are described in [157].

Figure 5.2: The semantics of and ports expressed using the ERL.

Figure 5.3: The semantics of xor ports expressed using the ERL.

Flows A flow in the PID is described by a predicate notation in the ERL called ERL flow. For example, a flow new_part carrying parameters n and p is written as
5.1. The Conceptual Level Coupling

Figure 5.4: The semantics of or ports expressed using the ERL.

Figure 5.5: The semantics of the cond ports expressed using the ERL.

new_part(n,p). It may link rules described by two processes, a process and an ERT view, a process and an external agent, or a process and a timer.

ERT views An ERT view is defined in terms of an ERT access expression in the ERL. This selection constitutes a part of the ERT and is a structural representation of the contents of the ERT view. A process may read from an ERT view (corresponding to a flow from an ERT view to the process) or write to an ERT view (corresponding to a flow from the process to an ERT view). This may be expressed by ERT access expressions and flows in the ERL:

- From ERT view: if ERT access expression then flow
- To ERT view: if flow then ERT access expression

Figure 5.6: The semantics of the rep ports expressed using the ERL.
Chapter 5. Combining Process Based and Rule Based Approaches

External Agents  An external agent may be represented by an ERL rule describing the objects which are passed from/to it in terms of ERT objects. This may be expressed by ERL flows:

- From external agent: when flow or if flow
- To external agent: then flow

Flows name tuples of data (single or aggregates) selected from the ERT and they allow information to be shared between a rule and an external agent [80].

Timers  The details of timers can be expressed using temporal constructs in the ERL. The temporal constructs may contain temporal connectives such as sometime.in_past and just.before, references to time points through predicates such as time.is and references to time intervals through functions such as today and this.week.

Instead of specifying all the details of the temporal formulae in the ERL, the notation of timers in the PID may be extended. This may be achieved by adding operators to the flows and define their connection to the ERL. By doing it in this manner, parts of the temporal formulae associated with a timer may be derived from PID specifications. One method for adding symbols to the flows was proposed in [80].

Rule Construction from a PID Specification

In the previous subsection, we defined the basis of how a process including ports can be described by a single rule expressed using the ERL. It may be realised that when the ports of processes are nested, the resulting rule structure may become very complicated. For practical purposes we need some method to reduce this complexity. Furthermore, we must deal with various levels of cooperating processes, that is, how rules are related to processes at various levels of decomposition. Thus there are at least three cases we have to deal with when constructing rules from a PID specification:

1. How nested input and output ports affect rule construction.
2. How the relationship between input flows and output flows affect rule construction.

Here we limit the description to major principles of rule construction. How the process of rule construction can be supported, is elaborated in Chapters 8.
1. Nested input ports and nested output ports. A process including nested input ports together with the resulting rules is shown in Figure 5.7. To construct the rules associated with this process, the trigger, condition, and action fields of a basic ERL rule are extracted from the process structure according to the definitions of the coupling in the Figures 5.2 to 5.6. The figures illustrate that a process with nested input ports does not need to be translated into one ERL rule, but to a set of ERL rules in which each rule may have composite trigger and condition fields. In a similar manner, a process with nested output ports may be translated into one ERL rule but a set of ERL rules in which each rule may have a composite action field.

![Diagram](image.png)

**Figure 5.7: Example of a process and its corresponding ERL rules.**

It should be noted that so far we have not taken the internals of the process into consideration. Thus, there is a certain ambiguity in the rules generated. From what we have already described, $\delta_1$ or $\delta_2$ will be in the action part of both rules. However, it is quite probable that $\delta_1$ is associated to only $\alpha_1$ and that $\delta_2$ is associated to only $\alpha_2$. To decide whether rules should have $\delta_1$ or $\delta_2$, $\delta_1$, or $\delta_2$ as their action part, the relationship between the inputs and outputs must be considered.

Before we elaborate on the relationships between flows and output flows, we will take a closer look at rule construction when a process has input or output flows that are xor related. A process with two input flows that are xor related is shown in Figure 5.8. This corresponds to a disjunction in the when part of a rule:

```
when $\alpha_1$ or $\alpha_2$ if $\gamma$ then ...
```

![Diagram](image.png)

**Figure 5.8: Example of a process with an XOR port.**

This may be represented in two separate rules instead of the disjunction in the when part:

```
when $\alpha_1$ if $\gamma_1$ then ...

when $\alpha_2$ if $\gamma_2$ then ...
```
In Figure 5.9 a process with four input flows is shown. In ERL this can be expressed as:

\[
\text{when } (\alpha_1 \text{ and } \beta_1) \text{ or } (\alpha_2 \text{ and } \beta_2) \text{ then ...}
\]

![Diagram of a process with an OR port.](image)

Figure 5.9: Example of a process with an OR port.

According to the above guideline for rule construction, this is most likely to be expressed as:

\[
\text{when } (\alpha_1 \text{ and } \beta_1) \text{ then ...}
\]
\[
\text{when } (\alpha_2 \text{ and } \beta_2) \text{ then ...}
\]

If disjunctions appear between non-triggering flows (corresponds to the if part of a rule) or between output flows (corresponds to the when part of rule), a composite rule can be split into simpler rules in the same manner as shown above.

2. Relationship between input flows and output flows  By taking into account the internals of a process, we may represent the way inputs and outputs of a process are related to each other. In Figure 5.10 the internals of the process in Figure 5.7 are taken into account. By doing this, the ambiguity in the two ERL rules as described above, is removed. Figure 5.10 also shows the simplified rules associated with the process.

![Diagram of a decomposed process and its corresponding ERL rules.](image)
3. Decomposed versus non-decomposed processes Each non-decomposed process should have associated a set of ERL rules describing the behaviour of the process. In addition, one may optionally specify the behaviour of decomposed processes, these rules being interpreted as constraints on the behaviour of the rules describing the non-decomposed processes. The rules associated with the processes at the lowest level of decomposition constitute the rule base which is mapped down to executable TEQUEL rules.

5.2 The Design Level Coupling

The coupling between the ERL and the PID at the conceptual level was defined with respect to the logical semantics of the two languages. At the design level, we will use the conceptual level coupling together with the executional semantics of the ERL as basis for defining the executional semantics of the PID.

Executable Specifications

In the previous section we defined the coupling between the PID and the ERL at the conceptual level, but this is insufficient to provide a formal definition of the dynamics of a system. The specifications from which executable rules shall be generated must be expressed using a form of the ERL rules that has precisely defined execution\(^1\) semantics. This means that by the time a set of ERL rules can be mapped to an executable TEQUEL form:

1. The rules must contain all the details of the system, e.g., unspecified predicates must be completed.
2. Each rule must be classified as an event-action rule, a derivation rule, or a constraint rule.
3. The rules must have been modified to fit with the computational model of the executable rules, i.e., the temporal model.

An automatic mapping from non-executable ERL rules to executable TEQUEL rules will be possible only if the above rules are met. An automatic mapping is essential if maintenance is to be made on ERL specifications rather than TEQUEL specifications (corresponding to code level). If the rules are not obeyed, manual interventions in the mapping process would be necessary.

\(^1\)A language has executional semantics if it is executable itself or it can be transformed into another representation which is executable, without manual intervention.
The Semantics of ERL Rule Types

The ERL is given executional semantics by an underlying formal temporal model and a mathematically rigorous definition is given in [94]. Here we provide an informal description of the executional semantics of the ERL. Based on this the executional semantics of the PID are described.

A temporal structure is introduced to describe temporal information included in a system specification. The temporal structure consists of a set of databases $\Delta_t$ where $t$ denotes the time associated with a particular database, as illustrated in Figure 5.11. Thus, the temporal structure gives us a means to describe how the data in the system evolves over time.

![Diagram](image)

**Figure 5.11: Temporal structure to describe temporal information.**

In Figure 5.11, each database corresponds to all information represented in the system, not only the contents of the physical databases. Therefore, $\Delta_t$ denotes both data in the databases and transient information in the system contained in flows at time $t$. The $\delta$ and $\delta'$ represent the set of changes made to the database in moving from one version $\Delta_{t-1}$ to another $\Delta_t$, as well as the most recent external input to the system. We will now present how the temporal structure can be used to define the semantics of ERL rules.

**Event-action rules** To describe the semantics of an event-action rule we only need to consider the set of changes $\delta$ which brought the system to the current database $\Delta_t$, the current database $\Delta_t$, and the set of changes $\delta'$ which will take the system to the next version of the database. The semantics of an event-action rule:

```
when trigger if condition then action
```

are that whenever `trigger` was part of the set of changes $\delta$, then provided `condition` is provable from the current database $\Delta_t$ the `action` must be part of the next set of
changes to the database $\delta'$. This is illustrated in Figure 5.12.

![Diagram](image)

**Figure 5.12: Relating rules and databases.**

The *condition* is provable from the current database $\Delta_t$. The definition of what it means for a trigger to be part of $\delta$ and an action to be part of a set of changes $\delta'$ is as follows:

- A trigger is part of $\delta$ if there is a member of $\delta$ which can be unified with the trigger. A trigger is either an insertion, deletion, or update to the database, an external or internal flow.

- An action is a temporal formula with the atoms being either insertions, deletions or updates to the database, external flows, or internal flows. The action is part of $\delta'$ if there is a member of $\delta'$ which performs the appropriate insertion, deletion, or update to the database or produces the appropriate internal or external flow.

From the above we can conclude that a new database $\Delta_{t+1}$ can be derived from the current database $\Delta_t$, the most recent set of changes $\delta$, and a single action rule. It should be noted that in the general case the action of the rule need not be made in $\delta'$, but could be scheduled to take place in any future transition $\delta', \delta'', \delta''', \ldots$. However, for the TEMPORA project only the simple case is considered. Actions in the ERL may update the historical database model held in $\Delta_t$ to change information in the model about the past, present, or the future.

There are also some other points that can be made about the semantics above that concerns the practical use of the ERL. First, the definitions above are limited to yield one rule. In practice, of course, there are many rules in use, and so we must extend the above definitions to cope with multiple rules. We refer to [94] for a detailed description.

Second, the semantics above must be extended to deal with transactions. That is, mechanisms to hide updates of each transaction from other transactions, until that
transaction commits must be provided. While the model we use is rich enough to supply such mechanisms, it is beyond the scope of this thesis to present the details here. The interested reader is again referred to [94].

**Derivation rules and predicates** A derivation rule expresses that some fact holds if some set of other facts hold. The semantics of a derivation rule:

\[
\text{if } expression_1 \text{ then } expression_2
\]

are that whenever the \( expression_1 \) holds on the current database \( \Delta_t \) for a set of variable substitutions then \( expression_2 \) also holds for the same set of variable substitutions. Predicates are similar to derivation rules. However, a derivation rule is associated with a flow or an ERT relationship but a predicate is not.

**Constraint rules** A constraint rule specifies that some condition must not be violated. The semantics of a constraint rule:

\[
\text{if } expression_1 \text{ then } expression_2
\]

are that whenever the \( expression_1 \) holds on the current database \( \Delta_t \) for a set of variable substitutions then \( expression_2 \) also must hold for the same set of variable substitutions.

**The Executional Semantics of the PID**

The conceptual level coupling is used together with the executional semantics of the ERL as basis for defining the executional semantics of the PID. The semantics of relevant constructs of the PID (defined with respect to the temporal structure above):

- A triggering flow is part of \( \delta \) if there is a member of \( \delta \) which can be unified with the triggering flow. A process’ trigger is a flow, where we classify a flow as an *external flow* if from an external agent, an *internal flow* if from a timer or a process (i.e., internal flow), or an *database transition* if from an ERT view.

- A non-triggerring flow and the internals of the process are provable from the current database \( \Delta_t \). A non-triggerring flow may originate from an external agent, a process, or an ERT view.
5.3. Rule Execution in a Temporal Database Environment

- An output flow is part of $\delta'$ if there is a member of $\delta'$ which performs the appropriate insertion, deletion, or update to the database (i.e., the flow to an ERT view along with specification of parts of the ERT to be modified) or produces the appropriate internal or external flow. An output flow is a flow, where we classify a flow as an external flow if to an external agent, an internal flow if to a timer or a process (i.e., internal flow), or an database transition if to an ERT view.

The other constructs of the PID have no executional semantics.

There is at least one event-action rule associated to each process in the PID. However, there may be event-action rules which are not associated to any process. Derivation rules and constraint rules are not directly associated to a process. How the coupling between the various types of rules and the other languages can be made (i.e., between the ERL and the PID, and the ERL and the ERT) are further described in the next chapter.

5.3 Rule Execution in a Temporal Database Environment

To be executed, ERL rules at the Design Modelling Level are grouped together in transactions and mapped to executable TEQUEL rules. These rules are compatible with a rule manager which controls the execution of the system. The rule manager is part of the TEMPORA run-time system shown in Figure 3.1. Since the focus of the thesis is on the early stages of the systems development we will only illuminate three points of rule execution which are important for the understanding of the use of rules in TEMPORA: transactions, rule evaluation, and the use of a temporal database. The rule manager controls the execution of the rules and manages the temporal database.

Transactions

In TEMPORA, transactions have the usual database semantics which entails that they have the ACID properties [31, 146]. PID specifications are used as basis for design of transactions in TEMPORA. Transactions may be triggered by a flow from an external agent, timer, process, or an ERT view. The coupling between the PID and ERL is used to derive the associated event-action rules to be placed in a transaction. The duration of a process is decided by the number of clock ticks that the associated rules take to execute.
Chapter 5. Combining Process Based and Rule Based Approaches

Rule Evaluation

The work on rule evaluation in TEMPORA is based on work previously done on executable temporal logics, e.g., USF [41] and MetateM [8]. In that work, a declarative view of rule evaluation is taken. A set of rules of the form

\[ \text{formula about the past } \Rightarrow \text{formula about the future} \]

are evaluated with respect to a particular state in a temporal database, yielding a number of formulae about the future which must be made true, if they are not already true [41]. In principle, it is possible to modify ERL rules to this "past ⇒ future" normal form. In this view, actions never update the past and thus, the history of the past is kept. However, this idealised "past ⇒ future" normal form is modified in TEMPORA. For pragmatic reasons, we allow actions to be updated on the past and queries to be made on the future, i.e., allow both queries and actions to the past, present, and current. This means that the conditions of rules may be modified and we cannot go back in time and find the complete history of the database, as is the case in a pure deductive database.

Using a Temporal Database

The executional semantics of the ERL and the PID as described above can be used with any database whether relational, object-oriented, deductive, or another type, since the temporal structure was defined for complete databases. In TEMPORA, the rule manager uses a temporal database based on the SYBASE RDBMS.

Using a temporal database implies two flows of time [119, 130]:

- The event time (also called valid time [130, 61]), which records as a series of time points \( E = [e_1, e_2, \ldots] \) the time over which we know a piece of information holds in the Universe of Discourse (UoD).

- The transaction time, which records as a series of time points \( T = [t_1, t_2, \ldots] \) the time over which the information holds in the information system.

The first is the flow of time which is modelled within each version of the database. Each individual version of the database can be imagined as a sequence of states. In TEMPORA, as in many other temporal database systems, the flow of time is described as a series of snapshots over a bounded, discrete, linear flow of time. The second is the flow of time along which the database changes, i.e., tracks the database evolution. The two flows of time are illustrated in Figure 5.13, adopted from [35]. Querying using the ERL works along the event time axis. The when part of rules detects changes along the transaction time axis. Execution of ERL rules that have
5.3. Rule Execution in a Temporal Database Environment

![Time Flow Diagram](image)

**Figure 5.13: Two flows of time: event time and transaction time.**

Updates of the database in their then part causes updates to take place along the transaction time axis.

At any given time, there is a distinguished state in the sequence known as the current state, which queries are evaluated with respect to. Queries can contain references to historical information, and relate previous values of information to current values. Languages for implementing ERL-like derivation rules have been devised by Abadi and Manna [1]; the semantics of such languages are well understood.

To illuminate the difference between event time and transaction time, we will elaborate on some aspects of the rule execution and give some examples.

**Transaction time and execution time** Transaction time depends on the execution time, and is independent of when things are recorded as having happened, i.e., the event time associated with the data. Except that the “current-time” used for event time queries is kept as close to the execution time as possible. For example, if something happened to occur before 1 am and that is executed and something else is going to be triggered at 1 am, one is not sure if that rule will be executed at 1 am, cf., Figure 5.13. It may be executed later on because of delays in the execution. However, the event time used in the database query will be 1 am, and thus the query is executed as if it had been executed at the transaction time at which it was made.

**Queries and actions** Queries are always on event time. A query on the event time at transaction time $t$, i.e., current database $\Delta_t$, is carried out on the current database. In contrast, an action (e.g., an update) to event time at transaction time $t$ is at earliest executed at transaction time $t+1$, i.e., new database $\Delta_{t+1}$.

\(^2\)Complete in the sense that everything we know is included in the database.
Asynchronous execution of transactions Transactions are executed asynchronously in TEMPORA. That means that if two flows between transactions occur at the same time they are not necessarily executed at the same time, i.e., transaction time. That depends on what may already be executed of the system load, i.e., how much the system has to do. However, within a transaction, the CVE loop is used [135].

5.4 Chapter Summary

This chapter has described the coupling between process based and rule based approaches both at the conceptual and design levels. At the conceptual level the coupling was defined using the logical semantics of the languages. Based on this coupling we presented a graphical representation of rules and described how the elements of a PID can be expressed using the ERL. At the design level the coupling between the ERL and the PID was made via the executional semantics of the ERL together with the coupling at the conceptual level. In this way, we have given an executional semantics to the PID via the underlying temporal model of the ERL.

The rules at the design level are grouped together in transactions and mapped to executable TEQUEL rules in order to be executed. An automatical mapping from non-executable ERL rules to executable TEQUEL rules is essential if maintenance is to be made on ERL specifications rather than TEQUEL specifications (corresponding to code level). The coupling between the PID and the ERL provides us with facilities to do transaction design using PID specifications as the basis. In the next chapters we will show that the coupling is also essential for supporting complexity reduction when the TEMPORA languages are used. Via the coupling we can utilise the abstraction/structuring and visual features of the PID to enhance rule modelling in information systems development.

Transaction design and rule execution as such are beyond the scope of this text. In this chapter, we have only included descriptions about these when necessary to illuminate central issues of rules in different facets of the modelling process.
Chapter 6

Structuring Mechanisms

This chapter describes a set of structuring mechanisms for the TEMPORA language.

Previous work which has been carried out on structuring mechanisms in the TEMPORA project has been a valuable basis for the work in this thesis. In particular, the work on rule relations by G. Sindre [116] provided useful input. The rule relations together with experiences using them in practice, have already been given in Chapter 3 and will not be repeated here. The previous work has been modified and extended in the following directions:

- A general framework for structuring mechanisms is proposed and general classes of links are formally defined.
- A revised version of rule relations is suggested and formally defined.
- Various links between the ERT, the ERL and the PID are proposed and formally defined.

Section 6.1 presents a framework for structuring mechanisms. Basic concepts are defined and a classification of structuring mechanisms is proposed. Section 6.2 gives an overview of various mechanisms provided in TEMPORA for structuring a conceptual specification. A more detailed description of the various types of structuring mechanisms together with examples is then given in Sections 6.3, 6.4, and 6.5.

6.1 A Framework for Structuring Mechanisms

The Purpose of Structuring Mechanisms

Large and complex real world problems call for adequate structuring mechanisms to deal with the large amounts of detail they contain. This is in contrast to most 'toy
problems' where all the details can be dealt with at the same level. The purpose of structuring mechanisms varies at different stages of development. We distinguish between structuring mechanisms used in the early phases of development, the specification level and structuring mechanisms used by the executorial system, the execution level. Since the focus of this work is on the early stages of development, the problem of structuring at the specification level will be emphasised.

**Structuring mechanisms at the specification level** The main objective of structuring mechanisms at the specification level is *to improve the communication and understanding aspects of specifications*. They should provide the means to relate domain knowledge at different levels of detail in a systematic manner, i.e., they provide the 'glue' to connect related components of specifications. To achieve this, the structuring mechanisms must provide means to (Figure 6.1):

1. Consolidate the various languages that are used in modelling to enable the generation of integrated systems specifications—connectivity.

2. Facilitate grouping of details into desirable knowledge chunks—modularity.

![Figure 6.1: Structuring mechanisms ensure integrated specifications.](image)

The first point implies that the structuring mechanisms should contribute *to bridge the gap between the various specification languages* and thereby reduce the fragmentation of specifications when expressed in several languages. A thorough understanding of both the languages and how these are related, is necessary. For instance, similar and associated concepts in the languages must be identified to deal with redundancy in the specifications. Figure 6.1a shows a situation where such links do not exist whereas Figure 6.1 shows a situation where related concepts are linked. The second point, grouping, implies that facilities to structure details into more comprehensible units are provided.
Structuring mechanisms at the execution level  The main objective of structuring mechanisms at the execution level is to increase the performance of the executable system and facilitate tracing of specifications. It may involve restructuring of database schemas (e.g., denormalisation), reducing the number of rules, techniques for pruning the solution space (e.g., applying clustering of rules/clustering hierarchies for pruning the solution space), etc.

By providing explicit links between the specification level and the execution level, it is possible to see how expressions at higher levels are reflected in the executable systems specifications, e.g., how rules at the specification level are implemented. This would also allow the semantics of the system specified at the specification level to be exploited at the execution level.

Types of Structuring Mechanisms

Structuring mechanisms provide the links which allow for structuring of pieces of information which are in someway related. They can take a variety of forms such as modelling constructs of a language, definitions of couplings between languages and special facilities that are built in tools. The links may connect related information within a model as well as across models located within the same specification level or at different levels. We classify structuring mechanisms into three major classes (Figure 6.2): intra-language links, inter-language links and inter-level links. Each type differs in either type of specification or level of specification. By distinguishing between level and type we could have had four different types but we have chosen to threat the types which differ in specification levels together. As we will show later in this chapter, we are mostly interested in keeping track of how a specification expressed using one language evolves during the development process.

![Figure 6.2: Possible types of links between specification components.](image)

We will now formally define the various types of structuring mechanisms. First, we
present the basis for describing the various links. A specification \( S \) can be identified by the pair:

\[
S = (T, L)
\]

where \( T \) denotes type of specification and \( L \) denotes level of specification.

Examples using the TEMPORA languages:

\[
T \in \{ \text{ERL, ERT, PID} \}
\]

\[
L \in \{ \text{Business Modelling Level, Information Systems Modelling Level, Design Modelling Level, Run Time Level} \}
\]

\[
S_1 = (\text{ERT, Business Modelling Level})
\]

\[
S_2 = (\text{ERT, Information Systems Modelling Level})
\]

\[
S_3 = (\text{PID, Business Modelling Level})
\]

To give a proper definition of a link between two specifications, we are not only interested in that there is a link between them but also what components in the specifications that are connected by the particular link. To achieve this the definition of a link must be extended as follows. Rather than saying a link connects two specification we may say that a link \( Rel \) can be described by the pair of components it connects:

\[
Rel = \{ \langle O_i, S_i \rangle , \langle O_j, S_j \rangle \}
\]

where

- \( O_i \) and \( O_j \) denote object types contained in the specifications \( S_i \) and \( S_j \), respectively.

In this way we can explicitly specify what components in the specifications that are related by the link. Examples using the TEMPORA languages:

\[
O \in \{ \text{Entity classes, Relationship classes, Value classes, Complex entity classes, Processes, ERT views, External agents, Timers, Flows, Ports} \}
\]

\[
T \in \{ \text{ERT, PID} \}
\]

\[
L \in \{ \text{Business Modelling Level, Information Systems Modelling Level} \}
\]

The formal basis above will be used when the various types of links are elaborated for the TEMPORA languages in Sections 6.3 to 6.5.
6.1. A Framework for Structuring Mechanisms

**Intra-language links** Structuring mechanisms targeted at a specific language are called *intra-language links*. They include means for structuring information within a set of specifications expressed using one language. By using the definitions given above, an intra-language link can formally be described as follows:

\[ Rel_{\text{intra-language}} = \langle \langle O_i, \langle T_i, L \rangle \rangle, \langle O_j, \langle T_i, L \rangle \rangle \rangle \]

I.e., links between specifications of the same type and located at the same specification level.

Every conceptual language is based on an assumption that certain concepts are more important than others. Accordingly, these core concepts may form the basic structure of specifications expressed using the language, e.g., objects in object-oriented approaches. Hence, we distinguish between constructs of a language that have structuring and non-structuring properties by whether the construct \( O \), appears in any \( Rel_{\text{intra-language}} \).

**Inter-language links** Structuring mechanisms targeted at relating two specifications expressed using different languages to each other are called *inter-language links*. They include means for structuring information between specifications at the same specification level. The specifications are expressed using different but related languages. By using the definitions given above, an inter-language link can formally be described as follows:

\[ Rel_{\text{inter-language}} = \langle \langle O_i, \langle T_i, L \rangle \rangle, \langle O_j, \langle T_j, L \rangle \rangle \rangle \]

where

- \( T_i \neq T_j \): the languages used for the specifications are different.
- The specifications are located at the same specification level.

Inter-language links contribute to bridge the gap between languages by relating corresponding or associated concepts in the different languages. An example of inter-language links is the coupling between the PID and the ERL defined in the previous chapter.

**Inter-level links** Structuring mechanisms targeted at relating specifications at different specification levels to each other are called *inter-level links*. They include means for structuring information between specifications at different levels expressed using the same language or expressed using different languages. By using the definitions given above, an inter-level link can formally be described as follows:
\[ \text{Rel}_{\text{inter-level}} = \langle \langle O_i, \langle T_i, L_i \rangle \rangle, \langle O_j, \langle T_j, L_j \rangle \rangle \rangle \]

where

- \( L_i \neq L_j \): the specifications are located at different specification levels.
- The languages may or may not be the same.

Inter-level links serve the purpose of bridging the gap between languages used at different specification levels. These are often links between specifications expressed using the same languages (i.e., \( T_i \neq T_j \)). An example of a link between specifications located at different levels are relations between rules at different specification levels.

### 6.2 Structuring Mechanisms in TEMPORA - An Overview

During the work on the thesis, structuring mechanisms for the TEMPORA languages have continuously evolved. The work presented here has been updated by experience gained from using intermediate versions in practice. Experiences from early case studies revealed that there was a need to improve the baseline modelling languages' ability to deal with large and complex real world problems, which are characterised by containing massive amounts of information [100]. In particular, a tighter coupling between the TEMPORA languages and mechanisms to impose structure on the flat rule base were needed [100].

In the remainder of this chapter we will present different types of structuring mechanisms for specifications developed using the TEMPORA languages. Figure 6.3 gives an overview of the various links at the specification level for the TEMPORA languages. The specification level in TEMPORA comprises Business Modelling Level, Information Systems Modelling Level and Design Modelling Level. The execution level in TEMPORA is called the Run-time Level.

### 6.3 Intra-Language Links

The three specification languages in TEMPORA differ strongly when it comes to what extent structuring mechanisms are provided. As described in Chapter 3, the ERT provides the following structuring mechanisms: generalisation/specialisation (is-a hierarchies) and aggregation (complex entity classes). The PID provides structuring around processes by allowing decomposition of processes. Case studies (e.g., [100]) have shown that these inherent intra-language links of ERT and PID are sufficient
for modelling large industrial-sized applications. But as mentioned above, case studies also revealed that there was a strong demand for providing adequate mechanisms for structuring rules in order to cope with large rule bases [100]. Therefore in the remainder of this section, we only present intra-language links targeted at relating rules in a rule base. Such structuring mechanisms are called inter-rule links.

Revised Version of the Inter-Rule Links

Rules may be related to other rules in a number of ways: goal related, causal related, domain related, context related, exception related, etc. Which relations are to be included in a specification may differ in domains and applications.

Our suggestion is to propose a set of relations where each relation is associated with a set of features. These features can be used in the modelling process to navigate in the network of specifications and components of specifications. Here we will define
a set of relations that are domain and application independent. Inspired by previous work in TEMPORA [116, 156, 135, 102], related approaches in AI [7] and experiences using the initial version of the inter-rule links\(^1\), we propose a revised set of relations to relate rules in a rule base. The basic structure of an inter-rule link, i.e., a relation:

\[
\langle \text{rule} \rangle \langle \text{relation} \rangle \langle \text{rule} \rangle
\]

The rule fields may be any ERL expression and may represent an organisational goal, policy, action, plan, etc. The relation field may be any of the following relations: causes, uses, has.info.derived.by, constrained.by, relates.to, suspends, overrules, or their inverse. Table 6.1 provides an overview of the relations and their inverse together with an informal description of their semantics.

<table>
<thead>
<tr>
<th>relation</th>
<th>inverse</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>causes</td>
<td>caused.by</td>
<td>Causal relationship between two rules.</td>
</tr>
<tr>
<td>uses</td>
<td>used.by</td>
<td>Links a rule and a predicate used by that rule.</td>
</tr>
<tr>
<td>has.info.derived.by</td>
<td>derives.info.for</td>
<td>Links a rule and a derivation rule that derives information for that rule.</td>
</tr>
<tr>
<td>constrained.by</td>
<td>constrains</td>
<td>Links a rule and a constraint rule that defines constraints for that rule.</td>
</tr>
<tr>
<td>relates.to</td>
<td>is.related.to</td>
<td>Defines that two rules are related to each other in some way.</td>
</tr>
<tr>
<td>suspends</td>
<td>is.suspended.by</td>
<td>A rule suspends the execution of another rule.</td>
</tr>
<tr>
<td>overrules</td>
<td>is.overruled.by</td>
<td>A rule discards the execution of another rule.</td>
</tr>
</tbody>
</table>

Table 6.1: Overview of rule relations.

The relations causes, uses, has.info.derived.by and constrained.by can be derived from a TEMPORA specification expressed using the ERT, the ERL and the PID. The same is true for their inverse. However, the other relations relates.to, suspends and overrules must be manually entered and maintained. The inverse of these relations can always be derived.

We will now formally define the various types of relations. First, we present the basis for describing formally the various relations. A rule relation \( Rel_{\text{inter-rule}} \) can be described by the pair of ERL rules it connects:

\[
Rel_{\text{inter-rule}} = \langle R_i, R_j \rangle
\]

where \( R_i \) and \( R_j \) denote ERL rules.

An ERL rule \( R \) can be described by the tuple:

---

\(^1\)The inter-rule links proposed in the early stages of the TEMPORA project were presented in Chapter 3. Experiences using intermediate versions of these relations on examples and in case studies were also reported.
6.3. Intra-Language Links

\[ R = (\alpha, \beta, \delta, \tau) \]

where \( \alpha, \beta \) and \( \delta \) correspond to the \textbf{when}, \textbf{if} and \textbf{then} parts of an ERL rule, respectively. \( \tau \) describes the type of rule:

\[ \tau_i \in \{ \text{Action\_rule, Constraint\_rule, Derivation\_rule, Informal\_rule, Predicate} \} \]

We want to be able to refer to parts of a logic formula using the usual set operators such as union and intersection. This can be done by providing a function \( T \) that provides the mapping from a logic formula to a set of terms, designed as follows:

\[ T : \text{logic\_formula} \rightarrow \text{Set\_of\_Terms} \]

\[ T (A \text{ and } B) = T (A) \cup T (B) \]
\[ T (A \text{ or } B) = T (A) \cup T (B) \]
\[ T \text{ not } A = T (A) \]
\[ T (A) = \{ A \} \text{ where } A \text{ is a term} \]
\[ T (A) + \text{entity} (A) \rightarrow A, \text{ where } A \text{ is an entity} \]
\[ T (A) + \text{value} (A) \rightarrow A, \text{ where } A \text{ is a value} \]
\[ T (A \text{ rel } B) \rightarrow T (A) \cup T (B) \]
\[ T (A \text{ [rel } B, \ldots]) \rightarrow T (A \text{ rel } B) \cup T (A \text{ rel } \ldots) \]

Example:

\[ P \in T \{ a \text{ and } b \text{ and } c \text{ or } d \text{ and } e \} \]

Using the definitions above this means:

\[ P \in \{ a, b, c, d, e \} \]

\( T \) also supports intersection of ERT access expressions.

The \textbf{causes relation} denotes a causal relationship and thus, corresponds to the fact that a term appears in the \textbf{then} part of an event-action rule and in the \textbf{when} part of another\(^2\). The inverse of \textbf{causes} is \textbf{caused\_by}. Formally, a \textbf{causes} relation can be described as follows:

\[ Rel_{\text{causes}} = \langle (\rightarrow, \rightarrow, \delta_i, \text{Action\_rule}), \langle \alpha_j, \rightarrow, \rightarrow, \text{Action\_rule} \rangle \rangle \]

\(^2\)The \text{when} part of a rule corresponds to triggering flows in the PID.
where

\[ T(\delta_i) \cap T(\alpha_j) \neq \emptyset \]

The *uses relation* links a rule and a predicate rule used by that rule. The inverse of *uses* is *used_by*. Formally, a *uses* relation can be described as follows:

\[ Rel_{uses} = \langle (\alpha, \beta, \delta, \_), (\_ \rightarrow P, \text{predicate}) \rangle \]

where

\[ P \in T(\alpha) \cup T(\beta) \cup T(\delta) \]

The *has_info_derived_by relation* links a rule \( R_i \) and a derivation rule \( R_j \). \( R_i \) contains expressions that are derived by \( R_j \). The inverse of *has_info_derived_by* is *derives_info_for*. Formally, a *has_info_derived_by* relation can be described as follows:

\[ Rel_{has_info_derived_by} = \langle (\alpha, \beta, \delta, \_), (\_ \rightarrow D, \text{derivation_rule}) \rangle \]

where

\[ D \text{ has the form } A \text{ rel } B \land D \in T(\alpha) \cup T(\beta) \cup T(\delta) \]

The *constrained_by relation* links a rule \( R_i \) and a constraint rule \( R_j \) that defines a constraint on any of the expressions contained in \( R_i \). The inverse of *constrained_by* is *constrains*. Formally, a *constrained_by* relation can be described as follows:

\[ Rel_{constrained_by} = \langle (\alpha, \beta, \delta, \text{action_rule}), (\_ \rightarrow C, \text{constraint_rule}) \rangle \]

where

\[ T(C) \cap T(\alpha) \neq \emptyset \lor T(C) \cap T(\beta) \neq \emptyset \lor T(C) \cap T(\delta) \neq \emptyset \]
6.3. Intra-Language Links

The *relates_to* relation denotes that two rules are related to each other in some way which is intuitive to the user. Formally, a *relates_to* relation can be described as follows:

\[ \text{Rel}_{\text{relates_to}} = (R_i, R_j) \]

where \( R_i \neq R_j \). This means that *relates_to* may connect any pair of ERL rules. The inverse of *relates_to* is *is_related_to*.

The flexibility offered by this relation make it possible to link higher-level rules to clusters of lower-level rules. Furthermore, by associating an attribute to the relation specifying the criterion upon which the rules are related, the relation can be used to denote a variety of relationships between rules. This relation could also have been extended to a specific relation for each criterion, as is part is the case for the relations proposed by G. Sindre [116], e.g., the *motivates* and *necessitates* relations. However, we choose to give only one relation since such relations need to be manually entered and as reported in Chapter 3, they are difficult to maintain as more details are entered during the development process. Thus, it is important to keep the maintenance of such relations as simple as possible. By expressing such relationships by one relation, we are able to specify that two rules were somewhat related without further specification. The case studies showed that this was the main purpose of such links. Although case studies also showed it would be useful to have them specified in detail in separate relations, it turned out to be too time consuming to manually maintain them in the later stages of the development process.

To illustrate the use of attributes associated with the *relates_to* relation we provide an example. If we want to have a *motivate_domain* relation that denotes the fact that the existence of one rule is a consequence of the fact that the other holds. This can be expressed using the *relates_to* as follows:

\[ \text{Rel}_{\text{relates_to}} (\text{motivated_by}) \]

An example of another attribute is: *motivates_modelling_of*. This means that one rule motivated the capture of the other rule which is useful for recording the design rationale. We may also add other attributes to give more details about the relations if this is desirable. As mentioned before, the effort to maintain this information may be too time consuming. Thus, this is not pursued any further here.

The overrules and suspends relations The *overrules* and *suspends* relations and their inverse *overruled_by* and *is_suspended_by*, respectively, may be kept to specify explicitly priority between rules as defined by G. Sindre [116].
6.4 Inter-Language Links

Structuring mechanisms targeted at relating two or more specifications to each other are called *inter-language links*. Inter-language links contribute to bridge the gap between specifications by relating corresponding or associated concepts in different specifications. By having three languages in TEMPORA, we have the following categories inter-language links:

- ERT-ERL links
- ERL-PID links
- ERT-PID links

Inter-language links may differ according to how tight the integration between various specifications is. For each type of link, there may also be alternatives for how two specifications can be related. Here we present the different inter-language links that fit with the categories listed above, and for some of them we also outline alternatives for how the links can be realised. We use the basis developed in Section 6.1 for formally defining the links between objects located in different specifications. The general structure of an inter-language link $Rel_{\text{inter-language}}$ between objects in located in two different specifications $S_i$ and $S_j$ can be described by the pair of objects it connects:

$$Rel_{\text{inter-language}} = \{(\langle O_i, \langle T_i, L_i \rangle \rangle, \langle O_j, \langle T_j, L_j \rangle \rangle)\}$$

**ERT-ERL Links**

The ERT and the ERL can be related by providing links between objects (i.e., entity class, relationship class, or value class) used in the rules and the corresponding objects in the ERT. All rules, static as well as dynamic, are related to the ERT specification via the ERT objects they affect. Static integrity rules are related to the objects in the ERT that are constrained by the rules and static derivation rules are related to the derived entity classes and derived relationship classes in the ERT. In a similar manner, dynamic rules are related to the objects in the ERT they refer to/operate upon.

The $\text{rel}_\text{ert-to}$ is the link between ERT and ERL. It can be defined formally as follows: A link $Rel_{\text{refers-to}}^{\text{ERT-ERL}}$ between an object in an ERT specification $S_i$ and an object in an ERL specification $S_j$ can be described by the pair of objects it connects:

$$Rel_{\text{refers-to}}^{\text{ERT-ERL}} = \{(\langle O_i, \langle T_i, L_i \rangle \rangle, \langle O_j, \langle T_j, L_j \rangle \rangle)\}$$
where

- $T_i = ERT$ and $T_j = ERL$
- $L_i = L_j$
- $O_i \in \{\text{Entity\_class, Complex\_entity\_class, Derived\_entity\_class, Timestamped\_entity\_class, Relationship\_class, Derived\_relationship\_class, Timestamped\_relationship\_class, Value\_class}\}$. Here each of the arguments are a pair $\langle Name, Type \rangle$ but by abuse of notation we refer to the instance of the class by referring to the type.
- $O_j = R_j$ where $R_j = \langle \alpha, \beta, \delta, \tau \rangle$ as defined in the previous section.

More precisely, the possible links between objects in ERT and ERL can be defined as follows:

$$\text{Rel}_{\text{refers\_to}}^{\text{ERT} \rightarrow \text{ERL}} = \{ \langle \text{Entity\_class, } \langle \text{ERT, } L \rangle \rangle, \langle \text{ERT\_Access\_Expression, } \langle \text{ERL, } L \rangle \rangle \}, \langle \text{Relationship\_class, } \langle \text{ERT, } L \rangle \rangle, \langle \text{ERT\_Access\_Expression, } \langle \text{ERL, } L \rangle \rangle \}, \langle \text{Value\_class, } \langle \text{ERT, } L \rangle \rangle, \langle \text{ERT\_Access\_Expression, } \langle \text{ERL, } L \rangle \rangle \}, \langle \text{Complex\_entity\_class, } \langle \text{ERT, } L \rangle \rangle, \langle \text{ERT\_Access\_Expression, } \langle \text{ERL, } L \rangle \rangle \}, \langle \text{Timestamped\_entity\_class, } \langle \text{ERT, } L \rangle \rangle, \langle \text{ERT\_Access\_Expression, } \langle \text{ERL, } L \rangle \rangle \}, \langle \text{Derived\_relationships, } \langle \text{ERT, } L \rangle \rangle, \langle \text{ERT\_Access\_Expression, } \langle \text{ERL, } L \rangle \rangle \}, \langle \text{Derived\_entity\_class, } \langle \text{ERT, } L \rangle \rangle, \langle \text{ERT\_Access\_Expression, } \langle \text{ERL, } L \rangle \rangle \}, \langle \text{Derived\_value\_class, } \langle \text{ERT, } L \rangle \rangle, \langle \text{ERT\_Access\_Expression, } \langle \text{ERL, } L \rangle \rangle \}$$

The inverse of the \text{refers\_to} relation is the \text{referred\_by} relation. By maintaining both the \text{refers\_to} relation and its inverse, i.e., keeping bi-directed links, it will be easy to find the definition of an object used in a rule and also to find what rules that use a particular object, e.g., easy to find the object’s constraints. Using the links, the rules may be divided into groups according to which objects they use. Some priority mechanism for grouping may be developed, e.g., least common object may be chosen for attachment or a rule may be related to every object it affects$^3$.

### ERL-PID Links

The details of how we combine the rule-based approach and the process-oriented approach in TEMPORA was described in the previous chapter. Based on this coupling we can define a number of explicit links between the two languages. Firstly, we know

$^3$An approach where the objects are considered the major concepts and other concepts such as rules and processes are grouped according to the objects is similar to an object-oriented approach.
that we can associate each process with one or more event-action rules that actually control and describe the behaviour of a process. It will be easy to find the rules which are related to a process in some way (i.e., a dynamic action rule or a dynamic constraint) and also to find what processes that a particular rule is related to. Using the links, the rules may be divided into groups according to which process they are associated with and thus, we have a powerful clustering mechanism for rules. In this way, the PID provides an overall structure to the ERL rules. It is also possible to translate automatically between overlapping parts in the PID and the ERL.

We introduce two relations between the PID and the ERL: refers_to and implements. Their inverse is referred_by and implemented_by, respectively. The refers_to relation denotes the link between a rule and a construct in the PID described by the rule. The implements relation denotes the link between a set of event-action rules that implement a PID process.

The refers_to can be defined formally as follows: A link $\text{Rel}_{\text{ refers_to }}^{\text{ ERL-PID}}$ between an ERL rule $S_i$ and an object in a PID specification $S_j$ can be described by the pair of objects it connects:

$$\text{Rel}_{\text{ refers_to }}^{\text{ ERL-PID}} = \{\langle O_i, \langle T_i, L_i \rangle \rangle, \langle O_j, \langle T_j, L_j \rangle \rangle\}$$

where

- $T_i = \text{ ERL}$ and $T_j = \text{ PID}$
- $L_i = L_j$
- $O_i = \langle \alpha, \beta, \delta, \tau \rangle$
- $O_j \in \{\text{ Process, ERT_view, External_agent, Timer}\}$. Here each of the arguments are a pair $\langle \text{Name, Type} \rangle$ but by abuse of notation we refer to the instance of the class by referring to the type.

More precisely, the possible links between ERL rules and objects in the PID can be defined as follows:

$$\text{Rel}_{\text{ refers_to }}^{\text{ ERL-PID}} = \{\langle \langle \langle \langle \langle \langle \langle \langle \langle \langle \langle \langle \langle \alpha, \beta, \delta, \tau_1 \rangle, \text{ ERL, L} \rangle, \langle \text{ Process, PID, L} \rangle \rangle, \langle \text{ ERL, L} \rangle, \langle \text{ ERT_view, PID, L} \rangle \rangle, \langle \text{ ERL, L} \rangle, \langle \text{ External_agent, PID, L} \rangle \rangle, \langle \text{ ERL, L} \rangle, \langle \text{ Timer, PID, L} \rangle \rangle \rangle \}$

where $\tau_1 = \text{ action_rule, } \tau_2 = \text{ action_rule, } \tau_3 = \text{ action_rule, } \tau_4 = \text{ action_rule}$

The implements can be defined formally as follows: A link $\text{Rel}_{\text{ implements }}^{\text{ ERL-PID}}$ between an ERL rule $S_i$ and an object in a PID specification $S_j$ can be described by the pair of objects it connects:
6.4. Inter-Language Links

\[ \text{Rel}^{\text{ERL-PID}}_{\text{implements}} = \{\langle O_i, \langle T_i, L_i \rangle \rangle, \langle O_j, \langle T_j, L_j \rangle \rangle \} \]

where

- \( T_i = \text{ERL} \) and \( T_j = \text{PID} \)
- \( L_i = L_j \)
- \( O_i = \text{Action rule} \)
- \( O_j = \text{Process} \)

Currently, the \text{implements} relation only describes action rules associated with a process. By using the rule relations defined in the previous section, it is also possible to find all the rules that govern or constrain processes. For instance, the \text{is\_context\_related\_to} relation finds all the rules that are context related to each other, i.e., related to the same process. The context is all rules related to a particular process including the rules associated with objects consumed by the process. Using the rule relations defined in the previous section, an \text{is\_context\_related\_to} relation can be described as follows:

\[ \text{Rel}_{\text{is\_context\_related\_to}} = \{ \text{Rel\_causes} \cup \text{Rel\_uses} \cup \text{Rel\_has\_info\_derived\_by} \cup \text{Rel\_constrained\_by} \} \]

This relation gives all the event-action rules, predicates, derivation rules and constraint rules associated with a process.

**ERT-PID Links**

The ERT and the PID may be related by providing links between the objects used in the PID and the corresponding objects in the ERT.

Firstly, there is an explicit link between an ERT-view and the ERT. Every ERT view may be defined in terms of a selection in the ERT using ERL. This selection constitutes a part of the ERT and is a structural representation of the contents of the ERT view.

Secondly, it may be desirable to be able to find the definition of an object used in a process and also to find what processes that use a particular object. One alternative is to provide bi-directional links between the objects carried by the flows and the corresponding objects in the ERT. From these links, we derive the objects consumed by the process. Another alternative is to provide bi-directional links between the process’ objects to the ERT model. In this case, the objects in the ERT model
consumed by the process can be derived from the rules associated with a process using the coupling between the PID and the ERL, i.e., the objects are derived via rules and their links to the ERT model.

Thirdly, there would also be inter-language links between the external agents and the ERT, to capture the relationship between ERT objects and the messages sent to/by external agents.

The refers_to is the link between the ERT and the PID. The inverse of the refers_to relation is the referred_by relation. The refers_to can be defined formally as follows: A link $\text{Rel}^{\text{ERT-PID}}_{\text{refers_to}}$ between an object in an ERT specification $S_i$ and an object in a PID specification $S_j$ can be described by the pair of objects it connects:

$$\text{Rel}^{\text{ERT-PID}}_{\text{refers_to}} = \{\langle O_i, (T_i, L_i) \rangle, \langle O_j, (T_j, L_j) \rangle \}$$

where

- $T_i = \text{ERT}$ and $T_j = \text{PID}$
- $L_i = L_j$
- $O_i \in \lbrace \text{Entity\_class}, \text{Complex\_entity\_class}, \text{Derived\_entity\_class}, \text{Relationship\_class}, \text{Derived\_relationship\_class} \rbrace$
- $O_j \in \lbrace \text{Process}, \text{ERT\_view} \rbrace$

The links between ERT and PID that can be derived from the other links and thus, we do not specify each possible link in detail:

$$\text{Rel}^{\text{implements}}_{\text{ERL-PID}} + \text{Rel}^{\text{inter-rule}}_{\text{ERT-ERL}} + \text{Rel}^{\text{refers-to}}_{\text{ERT-PID}} \rightarrow \text{Rel}^{\text{refers-to}}_{\text{ERT-PID}}$$

A link from a process to ERT object can be found from the link between Process and ERT rule and the link between ERL rule and ERT object.

It is possible to define a neighbourhood link based on the transitive closure of the other links:

$$\text{Rel}^{1}_{\text{neighbourhood}} (\langle O_i, S_i \rangle, \langle O_j, S_j \rangle) = \text{Rel}^{\text{any\_other}}_{\text{neighbourhood}} (\langle O_i, S_i \rangle, \langle O_j, S_j \rangle)$$

$$\text{Rel}^{2}_{\text{neighbourhood}} (\langle O_i, S_i \rangle, \langle O_j, S_j \rangle) = \text{Rel}^{\text{any\_other}}_{\text{neighbourhood}} (\langle O_i, S_i \rangle, \langle O_k, S_k \rangle) \land \text{Rel}^{\text{any\_other}}_{\text{neighbourhood}} (\langle O_k, S_k \rangle, \langle O_j, S_j \rangle)$$
6.5 Inter-Level Links

This section outlines how inter-level links can be included in the TEMPORA approach.

Links Between Specification Levels

The evolution of specifications is closely related to what systems life-cycle that is chosen, i.e., to the development discipline which details how systems development following the TEMPORA approach should be carried out. We have mentioned before that TEMPORA views the modelling process as comprising a set of major specification (modelling) levels (Figure 6.3): Business Modelling, Information Systems Modelling, Information Systems Design and Run-Time Generation. The focus of the modelling is to develop a conceptual model of the enterprise, to develop a conceptual model of the information system, to perform information system design and generate run-time system, respectively.

However, we do not go directly in one step from one level to the other. Within each such (course) level several other levels may also be distinguished. These correspond to the evolution of specifications during each major level. Within each such specification level there are a set of identifiable versions of full specifications. The way in which links between components at different specification levels are kept, differs according to how a model is developed. We distinguish between two main ways of extending a model across specification levels:

1. Extensions or modifications are specified with respect to an existing model. This means that a model at a different level is not specified in full or directly modified but new components are specified with respect to existing ones.

2. Extensions or modifications are specified on the model itself. This means that a copy of the model at one level is used as a basis for further development at the next level.

For both cases we may want to be able to freeze a particular version of a specification and record modelling/design decisions during the development process. This demands for a comprehensive versioning and configuration control facility to manage specifications within and across specification levels. Such a facility is already specified in [3, 140] and a description of the central features is provided in Appendix B.

Below we outline how specifications represented in the ERT, the ERL, and the PID, respectively, are related across the major specification levels. The inter-level links described below are a subset of the general inter-level links introduced in Section 6.1. For the inter-level links addressed here the following applies:
\[ \text{Rel}_{\text{inter-level}} = \langle \langle O_i, \langle T_i, L_i \rangle \rangle, \langle O_j, \langle T_j, L_j \rangle \rangle \rangle \]

where

- \((T_i = T_j)\): the specifications are represented in the same languages.
- \((L_i \neq L_j)\): the specifications are located at different specification levels.

The details of how various components of specifications represented in the TEMPORA languages should be related at different specification levels are described in [137]. For formalisation of the various links we refer the interested reader to the respective work that has addressed the relevant topics (references are provided together with the description below). The versioning and configuration control facility provides the necessary support to manage different versions of each specification and facilities to build and keep track of various system configurations.

**ERT** During Business Modelling the main aim is to develop an enterprise-wide ERT model. Only the main concepts of the enterprise are captured and details (e.g., decomposition of complex objects) are included in the model if the enterprise necessitates. All the constructs of the ERT language are used except from derived entity classes, derived value classes, derived relationships and timestamping. An ERT model for a particular information system should be defined with respect to a global model of the enterprise, i.e., the concepts relevant for the target information system at the Information System Modelling level should be defined according to the enterprise ERT. Thus, the enterprise ERT developed during Business Modelling forms the backbone of the ERT at the Information System Modelling level. The target system ERT may be modified and extended compared to the enterprise ERT. During Design Modelling, the target system ERT is further refined with detailed design descriptions. The extension/modification with respect to an existing model can be realised by defining an ERT algebra similar to the Entity-Relationship Algebra [96], the Phenomenon Algebra [11], or the ONE-R algebra [157]. The versioning system keeps track of the various versions are related within and across levels.

**ERL** During Business Modelling the main rules about enterprise are captured. At this level, most rules are expressed using natural language. Basically the general structure (i.e., WHEN-IF-THEN) is specified for some rules and the fields are expressed using informal text. If details are specified, the rules should be declarative in the sense of having logical semantics but not operational semantics. Thus, there should be no distinction between different rule types, e.g., constraints and action rules.

During Information System Modelling the rule base that results from Business Modelling is further refined and detailed rules about the target information system are
specified. During Design Modelling, these rules are further refined and classified into derivation, constraint and action rules. They are also extended with detailed design descriptions design information and computational semantics necessary in the eventual implementation.

At all levels, we may want to keep both textual and formal versions of rules. The higher-level rules can also be linked to clusters of lower-level rules using the inter-rule links, as described in the previous section. The versioning system keeps track of how the various versions of rules are related within and across levels and what versions valid configurations consist of.

**PID** A PID of the enterprise is developed during Business Modelling. No distinction is made between manual and automated parts of the information system. Based upon the PID developed during late Business Modelling, the boundary of automation is specified. This means that the parts of the models of the enterprise that will be implemented (or supported) by a computerised information system are determined. The relevant parts are copied and used as a basis for further modification/extension during Information System Modelling. The resulting process specification is further decomposed and details about the target information system included. The final version at Information System Modelling is used as a basis for Design Modelling in a similar manner as the resulting specification from Business Modelling is used as a basis for Information System Modelling. During Design Modelling the target system PID is further refined until it contains sufficient design descriptions. Details of the process logic is specified by the ERL, as will be elaborated upon in the next chapter. In a similar manner, as above the versioning system keeps track of the different versions and provide particular facilities to avoid a cascade of changes in versions of diagrams at higher level of decomposition when lower levels are updated.

### Mapping from Specification Level to Execution Level

It is important that models used at different levels are related and references or links between corresponding concepts at different levels are kept by the system. This is not only true for the models at different specification levels, but also applies to the connections between the conceptual specifications and the executable specifications. In addition, such links can be used for improving the performance of the final computerised system, e.g., tracing of specifications.

The mapping from the specification level to the execution level should largely be an automated procedure (Figure 6.4). The specifications that result from the Design Modelling may need to be optimised and eventually migrated to different production environments. Optimisations performed must be recorded. In this way, changes performed at higher levels can be reflected in the run-time system taking advantage of previous optimisation procedures.

Transaction boundaries are identified using PIDs describing the target system [141].
SPECIFICATION LEVEL

EXECUTION LEVEL

Figure 6.4: Links between specification and execution level.

The grouping of rules found in the PID is utilised during the mapping step from Design Modelling to the run-time level. Moreover, the mapping between rules at the specification level and rules at the execution level maintain what rules relate to each other at the two levels. Thus, TEMPORA run-time system can be used to animate the execution of rules at the conceptual level. During execution of such rules the developer can get insight into what behaviour the rules actually model. The consequences of a rule if its conditions hold can be inspected. This is useful for finding some general problems that can arise in large rule bases produced by several analysts (as was the case with the Sweden Post) such as identifying two or more conceptual rules which have the same operational semantics and the failure of any rule to hold for a specific event. The run-time model in TEMPORA together with how the semantics of the system specified at the higher levels can be exploited at the execution level, are further described in [135, 81, 138].

In general, the way in which the specifications that result from the Design Modelling should be translated to executable specifications needs further investigation. We have to look into how we can keep more of the structuring knowledge provided at the specification level in the mapping step. In the current approach this information is partly lost and what is left is a flat rulebase. Thus, we may need techniques for pruning the solution space, i.e., selection of rules for execution. Applying clustering of rules/clustering hierarchies for pruning the solution space is heavily addressed in the Expert Systems area, e.g., [7]. In our context, we may achieve this by utilising the clustering of rules provided at the specification level. However, how this can be done and at the same time take into account the efficiency aspect, has not yet been considered in detail.
6.6 Chapter Summary

This chapter has suggested a framework for structuring mechanisms. We have classified structuring mechanisms into three major classes according to the type of the components that they link: intra-language links, inter-language links and inter-level links. We have also described a set of intra-language and inter-language links for the TEMPORA languages. Except from the inter-level rule relations, inter-level links have only been superficially dealt with since there is already work that addresses such links.

We could also have classified the structuring mechanisms according to how they are realised and then distinguished between inherent and auxiliary links. The former means that the links are part of the languages and they can be explicit or implicit. The isa and complex entity class constructs in the ERT are examples of explicit inherent links. In this work, we have also added a set of implicit links which means that no constructs are provided in the language but their coupling may have impact on the semantics of the respective languages, e.g., the PID-ERL coupling described in the previous chapter. Moreover, we have suggested a set of auxiliary constructs which are not part of any of the languages, i.e., the rule relations.

The structuring mechanism at the specification level may not only have structuring properties but also contribute to the formal interpretation of a specification by stating something which cannot be derived from an existing specification. Such links cannot be removed from a specification without loss of domain knowledge. In our approach we have tried to reduce such links as much as possible because of the effort to maintain manual links. The suspends and overrules relations are the only links that contribute to the formal interpretation of a TEMPORA specification. They assign priority to a set of rules which are to be applied in a particular situation. The added information (which is based on domain knowledge) represented by such relations may decide which rules are to be activated. Removing some of these relations from a specification without adding the corresponding information using other constructs may leave the execution of the rules in an ad hoc manner. The other links such as the implicit links between specifications (e.g., links between ERT objects and rules and processes which use the objects) can be implemented in tools to show explicitly how specifications and specification components are related but they do not contribute to the formal interpretation of a specification. Providing explicit links between related parts of a specification makes it easier to find the rules at the specification level (and thereby improve the comprehensibility of specifications) as well as at the execution level (and thereby improve the efficiency of a running system).

The structuring mechanisms as described in this chapter serve two purposes. Firstly, they may be considered as a complexity reducing facility in their own right by reducing fragmentation of specifications. This is achieved by linking related components located within the same specification or in different specifications and by providing special mechanisms for grouping related information. Secondly, integrated specifications is a prerequisite to utilise the potential of the filtering mechanisms suggested in the next chapter; unless facilities are provided to link equal and related com-
ponents it is impossible to use the filters to generate desirable simplifications of specifications. How the structuring mechanisms can be utilised in the modelling process, either separately or together with the filtering mechanisms, is addressed in Chapter 8.
Chapter 7

Filtering Mechanisms

We have already introduced a set of mechanisms to structure specifications expressed using the TEMPORA languages. However, this is not sufficient to deal with large specifications in the modelling process because of the massive amount of detail contained in such specifications. In this chapter, we suggest a set of filtering mechanisms which can be used to generate simplified views of specifications called viewspecs. The filters proposed are used to generate a variety of viewspecs of specifications expressed using the TEMPORA languages. In the next chapter, we show how viewspecs can be used as a basis for presenting subsets of the specifications as well as a basis for entering new information into the specifications.

Section 7.1 presents a framework for filtering mechanisms. Basic concepts are defined and a classification of filtering mechanisms is proposed. An overview of various types of filters for the TEMPORA languages is provided in Section 7.2. Sections 7.3 to 7.5 give a detailed description of various filters for the PID, the ERT and the ERL, respectively.

7.1 Framework for Filtering Mechanisms

Basic Concepts

A viewspec is a subset of a model, which has been selected by the user. Each viewspec holds a particular perspective of a model and thus, the notion of viewspecs gives us a means to view the model from different perspectives.

A viewspec is represented by a language and is a particular type of specification. It will always be associated with another specification. The specification from which the viewspec is generated will be referred to as the originating specification or just specification if it is obvious from the context. The means to generate a particular
viewspec of a specification is called a filter. A specification which is not generated by a filter will be referred to as a full specification. A filter defines an abstraction of a specification. The filter specifies the criteria which should be used to suppress details from the originating specification\(^1\) to produce the desired viewspec. In general, the means to generate viewspecs from composite specifications will be referred to as filtering mechanisms.

Characterising Abstraction Levels

Information may be expressed at different levels of detail and we will refer to a specific level of detail as an abstraction level. As we will see below, an abstraction level is an abstract notion and it does not seem to be possible to attain a unique stratification of abstraction levels. Nor does it seem to be desirable to have a fixed stratification because it may be perceived as a straight-jacket for the developer. Therefore, we only suggest a framework that provides means to describe various abstraction levels, viewspecs and filters.

We may classify abstraction levels at least in two dimensions (Figure 7.1) according to [115]:

- **Specification level.**

- **Amount of detail** contained in a specification within each specification level.

![Figure 7.1: Framework for describing abstraction levels.](image)

The concepts of specification levels and inter-level links that provide the 'glue' between specifications at different specification levels were described in the previous chapter. Within each specification level there is a set of identifiable versions of full specifications. The notion of abstraction level denotes what level of detail such specifications might be viewed at. Thus, a model can be viewed from different perspectives by looking at specifications at different specification levels and by varying the amount of detail within each level. A specific specification at an abstraction level

---

\(^1\) An originating specification may be a viewspec as well as a full specification. This concept is further dealt with in the next chapter.
A is determined by the tuple \((X, Y)\), where \(X\) describes what specification levels to focus on and \(Y\) describes the type of details to be included in the specification at the particular specification level. For instance, a PID specification at an abstraction level \(A_{top}^{PID}\) may be located at the highest specification level showing all the top-level processes in the domain where all but control flows are left out.

From the above we can see that the notion of abstraction level say something about what we want to focus on in the domain/system and at what level of detail we want to examine the relevant aspects of the domain. All these factors have an impact on the comprehensibility of a specification. Which abstraction level is desirable may vary during the development process. For example, at one point we want to look at all details of a PID specification and at a different point a viewspec of the specification may be sufficient. We can easily imagine that the number of possible abstraction levels is infinite. Just by making any change to a specification (full specification or viewspec), the abstraction level of the specification changes. Therefore, a unique stratification of abstraction levels is only possible by putting strict constraints on the manner in which specifications can be manipulated. This is however, not desirable from a modelling point of view. What we need is to be able to generate viewspecs in a flexible manner and provide means to refer to full specifications and their associated viewspecs in a unique way. How we can provide means to refer to a specification at a particular abstraction level is elaborated in the next chapter.

### Characterising Viewspecs

In the previous chapter, we stated that a specification \(S\) can be described by the pair:

\[
S = (T, L)
\]

where \(T\) denotes type of specification and \(L\) denotes level of specification. We will now refer to the set of all possible viewspecs as \(C^*\). A viewspec can be divided into subviewspecs for one of the two reasons: 1) subviewspecs are expressed using different languages or 2) subviewspecs are located at different specification levels. Thus, a definition of a specification \(C^*\) that encompasses the notion of viewspecs and subviewspecs:

\[
C^* = \{C_{1,1}^* \cup C_{2,1}^* \cup ... \cup C_{m,n}^* \}
\]

where

- \(C_{i,j}^* = \{(T_i, L_j)\}\) for which \(i = 1..m, j = 1..n\)
- \( m \) is the number of different languages and \( n \) is the number of different specification levels.

Viewspecs can be classified as follows (Figure 7.2):

![Diagram](image)

Figure 7.2: Examples of different types of viewspecs: a) an intra-language viewspec, b) an inter-language viewspec and c) an inter-level viewspec.

- **Intra-language viewspecs.** The class of intra-language view specs is formed of those viewspecs that have a single type and specification level. Formally, this can be expressed as:

  \[
  C_{\text{intra-language}} = \{ (T, L) \}
  \]

  where the viewspec is expressed using one language and it is located at the same specification level. An example is shown in Figure 7.2a.

- **Inter-language viewspecs.** The class of inter-language view specs is formed of those viewspecs that have a single specification level but the subviewspecs are expressed using two or more specification languages. All the subviewspecs are located at the same specification level. Formally, this can be expressed as:

  \[
  C^*_{\text{inter-language}} \subseteq \{ (T_i, L) \mid i \in 1..m, \text{ where } m > 1 \}
  \]

  where \( m \) is the number of different languages. That is, the subviewspecs may be expressed using two or more different languages but they are located at the same specification level. Figure 7.2b shows an inter-language viewspec which consists of two subviewspecs expressed using a diagrammatic and a non-diagrammatic language, respectively.

- **Inter-level viewspecs.** The class of inter-level view specs is formed of those viewspecs that have subviewspecs that are located at two or more different specification levels. They may be expressed using the same or different specification languages. Formally, this can be expressed as:

  \[
  C^*_{\text{inter-level}} \subseteq \{ (T_i, L) \mid i \in 1..m \text{ and } j \in 1..n, \text{ where } m \geq 1 \text{ and } n > 1 \}
  \]
That is, subviewspecs may be expressed using one or more different languages and they may also be located at two or more different specification levels, i.e., at least two of the $L_i$s are different. Figure 7.2c shows an inter-level viewspec which consists of two subviewspecs expressed using a non-diagrammatic language. The subviewspecs are generated from specifications which are located at different specification levels.

In the above, we have ignored how the relevant details should be presented and focused on aspects that relate to abstraction level. To provide a proper description of a viewspec we also need to specify how details should be presented. This is elaborated upon in the next chapter. In the reminder of this chapter, we will focus filters to generate intra-language viewspecs. Both general features of such filters as well as specific filters for the TEMPORA languages are described. In the next chapter, we present how these filters can be used together with other complexity reducing techniques to generate inter-language and inter-level viewspecs.

Formal Basis for Defining Specific Filters

To distinguish between types of viewspecs and instances of viewspecs, we denote an instance of a viewspec as $V$. A filter $F$ is a mapping from a specification that contains all possible viewspecs $V^*$ to a viewspec $V$, where the mapping is an abstraction of $V^*$. Formally, this can be expressed as follows:

$$F: V^* \rightarrow V$$

For language $L$ this is equivalent to:

$$F: \{V_{L_1}^*, \ldots, V_{L_n}^*\} \rightarrow \{V_{L_1}, \ldots, V_{L_n}\}$$

where

- If $L$ is a composite language, $n$ is the number of sublanguages.
- Each $V_{L_i}^*$ and $V_{L_i}$ where $i = 1..n$, are tuples of sets. Each set in the tuple consists of instantiation of concept classes of a language.

This can be used to present the formal basis for describing filters for the TEMPORA languages. A filter $F$ is a mapping from a TEMPORA specification that contains the set of all possible viewspecs $V^*$ to a viewspec $V$, where the mapping is an abstraction of $V^*$. Formally, this can be expressed as follows:
\[ \mathcal{F}: \{V_{ERT}^*, V_{ERL}^*, V_{PID}^*\} \rightarrow \{V_{ERT}, V_{ERL}, V_{PID}\} \]

where

- \( V_{ERT}^* = \langle E_1, E_2, E_3, E_4, E_5, E_6 \rangle \) and
  \[ E_1 = \{\text{Entity\_class}\}, \text{Entity\_class} = \text{Entity\_name1}, \text{Entity\_name2}, \ldots, \]
  \[ E_2 = \{\text{Relationship\_class}\}, \text{Relationship\_class} = \]
  \[ \text{Relationship\_name1}, \text{Relationship\_name2}, \ldots, \]
  \[ E_3 = \{\text{Is\_a\_relationship}\}, \ldots, \]
  \[ E_4 = \{\text{Value\_class}\}, \ldots, \]
  \[ E_5 = \{\text{Complex\_entity\_class}\}, \]
  \[ E_6 = \{\text{Cardinality\_constraint}\}, \]

- \( V_{ERL}^* = \langle R \rangle, \langle R \rangle = \{(\alpha, \beta, \delta, \tau)\} \) where \(\alpha, \beta\) and \(\delta\) correspond to the when, if and then parts of an ERL rule, respectively and \(\tau\) describes the type of rule as defined in Chapter 6. \(\alpha, \beta\) and \(\delta\) may be any ERL expression.

- \( V_{PID}^* = \langle P_1, P_2, P_3, P_4, P_5, P_6 \rangle \) and
  \[ P_1 = \{\text{Process}\}, \]
  \[ P_2 = \{\text{ERT\_view}\}, \]
  \[ P_3 = \{\text{External\_agent}\}, \]
  \[ P_4 = \{\text{Timer}\}, \]
  \[ P_5 = \{\text{Flow}\}, \]
  \[ P_6 = \{\text{Port}\}, \]

- \( V \subseteq V^* \)

What we mean by \( V \subseteq V^* \) can be described as follows:

- \( V_{ERT} = \langle E_1', E_2', E_3', E_4', E_5', E_6' \rangle \)
- \( V_{ERL} = \langle R' \rangle, \langle R' \rangle = \{(\alpha', \beta', \delta', \tau)\} \)
- \( V_{PID}^* = \langle P_1', P_2', P_3', P_4', P_5', P_6' \rangle \)

where for all of the components of the specifications, there are subset related pairs of components \((K, K')\) such that \(K\) is a component of originating specification and \(K'\) is a corresponding component in the generated viewspec. For instance, the subset related pairs of a PID specification and an associated viewspec\(^2\) \((V_{PID} \subseteq V_{PID}^*)\):\(^2\)

\(^2\)It should be noted that we abuse the notation by referring to \(V\) as a subset of \(V^*\) since both \(V^*\) and \(V\) are tuples of sets.
\[ P'_1 \subseteq P_1, P'_2 \subseteq P_2, \text{ etc.} \]

Similarly, pairs of subset related components can be found for an ERT specification and an associated viewspec:

\[ E'_1 \subseteq E_1, E'_2 \subseteq E_2, \text{ etc.} \]

For a rule specification and an associated viewspec it means that the following is true for one or more rules:

\[ R' \subseteq R \]

In Sections 7.3 to 7.5, we will use this as a basis for defining different types of filters and for defining various filters for the TEMPORA languages.

**Classification of Filters**

We classify filters into two major groups:

- *Language/meta-model filters*. Language filters suppress details in a specification with respect to constructs of the language that is used for its representation. A particular construct of the language is the focus of the filtering process. Thus, the *meta-model* can be used to ensure that only valid constructs from the language are present in the viewspecs.

- *Model/specification filters*. A model filter is defined with respect to a particular model (or set of specifications) and is therefore *domain dependent*. They provide possibilities to focus on certain aspects of the domain by selecting components in a specification according to domain specific criteria.

Other relevant aspects of filters include:

- *Inclusiveness/exclusiveness*. A filter can be defined by specifying the components to be included in the viewspec or by specifying the components to be excluded in the viewspec. This is referred to as inclusiveness and exclusiveness properties of the viewspecs, respectively.

- *Determinism/non-determinism*. A filter is deterministic if the resulting viewspec of performing the filter on a specification S is the same each time, given that it operates on S each time. If the result is not predictable, the filter is non-deterministic.
Global/local effects. We distinguish between two cases: (1) The scope of effects is local if there is no effect of the filter beyond the specification upon which it operates, and (2) it is global if the scope of effect is beyond the specification upon which it operates. A serious problem with filters with global effects is how to propagate changes to affected specifications. As we will see in the reminder of this chapter, most filters have only local effects.

Projection/approximation. A filter may generate a viewspec which is a projection (Figure 7.3a) or an approximation of the originating specification (Figure 7.3b). An example of an approximation may be to remove two constraint rules from a collection of five rules.

![Figure 7.3: a) The viewspec is a projection of the originating specification whereas b) is an approximation of the originating specification.](image)

## 7.2 An Overview of Filters in TEMPORA

In the remainder of this chapter, we will define a set of filters which will be used as basis for building specifications using the TEMPORA languages. An overview of the various filters is shown in Table 7.1.

There are a number of ways to simplify a specification, each focusing on a different aspect of a domain. The choice of a particular set of filters and the classification of these are of course highly subjective. The proposal below is based on experience gained through case studies and examples applying the TEMPORA conceptual languages [100, 142] and the PPP language, and experience and requirements put forward by other researchers and practioners in the field [29, 77].

Some critics may say that it would be sufficient to give the possibility for having a user-defined filter, without giving any directions to which specification details that should be abstracted away. However, experience has shown that there is a need to go about filtering mechanisms in a systematical manner, so as to provide the user adequate tool support and thus to avoid chaos. This is in particular true when several simplications to the same specification are allowed. Once the user is familiar with the predefined filters, new filters may be created based on the experience already gained of using the intitial filters.
### Table 7.1: An overview of filters in TEMPORA.

<table>
<thead>
<tr>
<th>Language filter</th>
<th>Model filter</th>
<th>Inclusive/Exclusive</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Port</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>Flow</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>Control flow</td>
<td>Inclusive</td>
</tr>
<tr>
<td></td>
<td>External agents</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>ERT .view</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>Timer</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>Process overview</td>
<td>Inclusive</td>
</tr>
<tr>
<td>ERT</td>
<td>Entity class</td>
<td>Inclusive</td>
</tr>
<tr>
<td></td>
<td>Value class</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>Timestamping</td>
<td>Inclusive/Exclusive</td>
</tr>
<tr>
<td></td>
<td>Generalisation</td>
<td>Inclusive/Exclusive</td>
</tr>
<tr>
<td></td>
<td>Complex entity class</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>Derived information</td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td>Cardinality constraint</td>
<td>Inclusive</td>
</tr>
<tr>
<td>ERL</td>
<td>Rule relations</td>
<td>Inclusive</td>
</tr>
<tr>
<td></td>
<td>Subsets of rule relations</td>
<td>Inclusive</td>
</tr>
</tbody>
</table>

#### 7.3 PID Filters

This section suggests a set of filters to generate various viewspecs for PID specifications. Each viewspec is a simplification of the specification according to a set of criteria.

The list of filters that we define for the PID in this thesis is as follows:

- **Language filters:**
  - *Port filters* are exclusive and abstract ports away from specifications.
  - *Flow filters* are exclusive and remove flows.
  - *Control flow filters* are inclusive and keep control flow but abstract away all other components.
  - *ERT .view filters* are exclusive and remove all ERT views.
  - *External agent filters* are exclusive and remove all external agents.
  - *Timer filters* are exclusive and remove all timers.
  - *Process overview filters* provides an overview of the hierarchical relationships between processes.

- **Model filters:**
Component filters are inclusive and keep a selected process together with its immediate components.

We have chosen to divide filters into basic and auxiliary filters. Port and flow filters are the basic filters whereas the rest are considered to be auxiliary filters. The basic filters are considered fundamental to handle the complexity of the PID and should be implemented first. The auxiliary filters constitute a set of simplifications which seem to be advantageous but are not considered crucial for using the PID.

Layout considerations which are specific to a filter, are discussed together with the main description of the filter. General layout considerations, that is, how restructured viewspecs are to be dealt with in general are discussed separately in the next chapter.

To illustrate the various filters, the acquisition activity in the Library Case Study is used as example (Figure 7.4).

Port Filters

Performing a port filter removes the logical relations between input flows (and output flows, respectively) of processes. We distinguish between two types of port filters: partial port filters and total port filters.

Partial port filters Performing a partial port filter removes all input and output ports but the outermost ports for all processes in a diagram. To distinguish the resulting ports from other plain ports, the notion of abstracted ports is introduced. The notation for an abstracted port is a filled port symbol, i.e., the outermost port of all input and output ports are shaded (Figure 7.5). Except from suppressing details about ports, other components components are not influenced by the partial port filter, i.e., outermost ports are shaded whilst processes, flows, ERT views, external agents, and timers all remain.

Formally, a port filter $F_{port}$ can be expressed as follows:

$$F_{port} : V_{PID} \rightarrow V_{port}$$

where

$$V_{PID} = \langle \text{Processes}, \text{ERT.views}, \text{External.Agents}, \text{Timers}, \text{Flows}, \text{Ports} \rangle$$

$$V_{port} = \langle \text{Processes}, \text{ERT.views}, \text{External.Agents}, \text{Timers}, \text{Flows}, A(\text{Ports}) \rangle$$

$$A(\text{Ports}) = \{ P \mid P \in \text{Ports}, \neg \exists P' \in \text{Ports} \land P' \in \text{Ports} \}$$
Figure 7.4: PID specification of the acquisition activity.

Figure 7.5: PID specification where the ports are abstracted away.
where \( O(P_1, P_2) \) if \( P_1 \) and \( P_2 \) are ports of a process and \( P_1 \) is outside of \( P_2 \).

Only the outermost ports will come through the filter and be made visible as abstracted ports; the other parts will be removed by the filter.

It can be argued that the diagram shown in Figure 7.4 is rather messy and difficult to comprehend. By applying a port filter, the diagram becomes simpler as shown in Figure 7.6.

![Diagram](image)

**Figure 7.6:** The acquisition activity where regular ports are substituted by abstracted ports.

**Total port filters** The ports may impose certain requirements on the location of flows entering or leaving a process, that is, where the connection point between a flow and a process is located. Crossing flows may result. A more relaxed notation removes the port symbols from the specification and arrows may enter and leave a process.
at any edge (Figure 7.7). After abstracted the ports away from a specification, a cluttered diagram should be restructured to avoid crossing flows.

**Figure 7.7: PID specification where the ports are removed.**

By applying a total port filter, the diagram becomes simpler as shown in Figure 7.7. Figure 7.8 shows the PID specification of the acquisition activity in Figure 7.4 where a total port filter has been applied and then the diagram redrawn to reduce the number of crossing flows.

If a total port filter is performed (i.e., ports are removed and arrows may enter and leave a port at any edge) and duplicates\(^3\) of flows are removed, the resulting diagram may resemble a DFD except from the inclusion of control flows.

**Flow Filters**

When flows are collapsed to form more composite flows, the details of the flow components are abstracted away. Given a situation as depicted in Figure 7.9a, where a process P has \(n\) input flows \(\alpha_1, \alpha_2, \ldots, \alpha_n\) and one output flow \(\delta\). Assume that a complex flow \(\alpha\) can be constructed from the flow components \(\alpha_1, \alpha_2, \ldots, \alpha_n\). Accordingly, the input flows of process P can be collapsed into the composite input flow \(\delta\) as depicted in Figure 7.9b. This is the basic form of a flow filter. To distinguish an abstracted flow from its flow components the former is depicted by a dashed arrow. The same yields output flows as depicted in Figures 7.9c and 7.9d.

However, the basic form of a flow filter as described above is not sufficient. To deal with abstraction of flows in a systematical manner, there are at least five issues that need to be addressed:

- How to deal with abstraction of flows which carry different types of items, that is, information and material items?
- How to provide an appropriate naming schema for abstracted flows?

---

\(^3\)The concept of ports enforces a more detailed specification of the flows. Extra flows are introduced in the diagram if the same flow is involved in different port structures of the same process. When ports are removed a number of duplicated flows with the same information and/or material contents may result.
Figure 7.8: PID specification of the acquisition activity where ports are removed.

- How to deal with the graphical representation of abstracted flows?
- How to deal with abstracted flows that cross decomposition levels?
- How to deal with the representation of the ports that the flows use?

Flows which carry different types of items  The flows that are abstracted may carry information and/or material items, that is, \( \alpha_1, \alpha_2, ..., \alpha_n \) may have the same or different information content and/or material content. It might be argued if we should allow to collapse information flow and material flow or if these should be kept separate at all times. It seems to be reasonable to make this a user-decision and as we will see later, it is advantageous to allow information flow and material flow to be collapsed.
7.3. PID Filters

\[ \begin{align*}
\alpha_1 \\
\alpha_2 \\
\vdots \\
\alpha_n \\
\end{align*} \quad \text{a)} \quad \delta \\
\begin{align*}
\alpha_{\text{abs}} \\
\end{align*} \quad \text{b)} \\
\begin{align*}
\alpha \\
\end{align*} \quad \text{c)} \\
\begin{align*}
\delta_1 \\
\delta_2 \\
\vdots \\
\delta_n \\
\end{align*} \quad \text{d)} \\

Figure 7.9: a) Process P with n input flows, b) the input flows are abstracted into a composite flow, c) Process P with n output flows, and d) the output flows are abstracted into a composite flow.

Naming of abstracted flows  When it comes to naming of abstracted components such as composite flows, they could be given by the analyst, generated by the system, or a combination of both. For simplicity reasons, we will assume that the naming is performed by the analyst. The system should support the collapsing of flow attributes something which may be crucial to handle composite flows, e.g., experience gained in the Swedish Post Case Study [133].

Graphical representation of abstracted flows  Allowing composite flows to be constructed from simpler ones, may induce problems concerning the graphical representation of flows. We will now give some examples showing the formal problems of decomposition of flows and suggest some solutions.

A flow filter must be able to deal with various types of flows. We distinguish between three main types of flows:

1. Flows that have the same origin and destination.
2. Flows that have different origin but the same destination.
3. Flows that have the same origin but different destination.
4. Flows that had different origin and destination.

In the first case, we deal with flows that have the same origin and destination. Figure 7.10a shows an example where two flows \( \delta_1 \) and \( \delta_2 \) are carried by different flows between the same two processes. Following the schema above, an abstracted flow \( \delta_{\text{abs}} \) as depicted in Figure 7.10b results.
Collapsing input and output flows between the same two components should also be allowed, i.e., local communication between two components. For instance, if there are input and output flows going between a process and an external agent, the flows may be collapsed, e.g., between P4.6 and Donator in Figure 7.8. Figure 7.11 shows three different layout alternatives for this case: the dashed flows may be drawn parallel to each other, collapsed into one dashed bidirectional flow, or collapsed into one dashed bidirectional flow depicted by a line.

In the second case, we deal with flows that have the same origin but different destinations as shown in Figure 7.12a. It is a simple case but Figure 7.12b illustrates the main point, namely that we allow flow to be split into simpler flows. Similarly, the
third case is handled by allowing flows that have different origin but same destination to be collapsed into a composite flow, as illustrated in Figures 7.13a and 7.13b, respectively.

![Figure 7.12: Splitting flows with different destination.](image)

The fourth case is not dealt with here. Although it may appear useful for flows which follow almost the same route in the diagram, we have decided not to support
it until more experience is gained using the other flow filters. In addition, it will be a rather large effort to provide support for such a filter as indicated by other research efforts on similar problems [15, 88].

**Abstracted flows that cross decomposition levels**  We have already allowed flows to be split into simpler flows and simpler flows to be collapsed into composite flows. However, if the flows go across decomposition level a new issue arises because the scope of effect of the flow filter is not local to the specification anymore. The question is then: How can we ensure that the effects of a filter are propagated in a consistent manner to affected specifications? Figure 7.14a depicts an input flow $\alpha$ which carries items having different processes as destinations. Figure 7.14b depicts the output counterpart, where the items carried by output flow $\delta$ originate from different processes.

![Figure 7.14: Problems of parallel flow filters.](image)

This could be solved in one of the following ways:

- Exclude abstraction of flows where such situations as described above arise (Figure 7.15a). That is, the flows are duplicated and a separate flow is going to each component.

- Split the composite flow $\alpha$ into several branches, each branch going to a different process (Figure 7.15b).

- Add an extra input process (output process) and introduce three new flows $\alpha_1$, $\alpha_2$, and $\alpha_3$ (Figure 7.15c).

Current process-oriented approaches do not support filters and take the first approach. By having a separate flow makes the diagram more complicated and it is obviously not a good solution. Introducing new artificial processes as the third approach suggests is also not a good solution; the diagram is further cluttered and this is exactly the opposite of the intention of introducing the filter concept. Also it contradicts the principle of conceptual modelling - you introduce a process just to model the splitting of a flow. From this, the second approach seems to be the obvious approach to go for. Splitting flows should be allowed. If a complex flow $F$ (or any of its flow components) serves as input to different processes the flow may be required to split up into several branches, each branch going to a different process.
(Figure 7.16). Figure 7.17 shows the acquisition activity where all possible parallel flows are collapsed, some flows with same origin but different destination are collapsed and some flows with same destination but different origin are collapsed. The contents of the flow may be identical or different, that is, the branches of the flow may carry same or different material/information items. As we will discuss in the next chapter, most of task of collapsing flows must be performed manually because of the heavy computation of graph algorithms for such problems.

Figure 7.16: A flow with three different destinations is split into three branches.

**Formal definition of the flow filter**  We will now define formally the flow filter. It should be noted that the filter lets simple flows through, only flows of the types above are abstracted.

\[ F_{\text{flow}}: V^{*}_{PID} \rightarrow V_{\text{flow}} \]

where

\[ V^{*}_{PID} = \langle \text{Processes, ERT_views, External_agents, Timers, Flows, Ports} \rangle \]
Figure 7.17: PID specification which shows different ways of collapsing flows.

\[ V_{flow} = \langle \text{Processes, ERT.views, External.agents, Timers, A (Flows), Ports} \rangle \]

Flow = \langle Name.Flow, Origin, Destination \rangle

Flows is a set of Flow.

A (Flows) can be defined as follows:

\[ A (Flows) = \{ F \mid (F = \langle \text{Name.Flow, Origin, Destination} \rangle \land F \in \text{Flows} \land \neg \exists F' \text{ such that } F' \in \text{Flows} \land (F' = \langle \_ , \_ , \_ \rangle \lor F' = \langle \_ , \_ , \_ , \text{Destination})]) \lor (F_1 \in \text{Flows} \land F_2 \in \text{Flows} \land F_1 = \langle \_ , \_ , \text{Origin} \rangle \land F_2 = \langle \_ , \_ , \text{Origin, Destination} \rangle \land F = \langle \_ , \_ , \text{Origin, Destination} \rangle \land F_1 \neq F_2 \land F = \langle \_ , \text{Origin, Destination} \rangle \rangle \lor (O = \{ \text{Origin} \mid \langle \_ , \_ , \text{Origin, Destination} \rangle \in \text{Flows} \} \land |O| > 1 \land F = \langle \_ , \text{"Grouped", Destination} , O , \text{Destination} \rangle \}) \]
$\mathcal{N} (P)$ is a function to generate names given a set of strings as argument.

**Control Flow Filters**

Only control flows and components to which these are linked, are kept. All non-triggering flows and remaining isolated components are abstracted away and a specification comprising processes, ERT views, external agents, timers, and triggering flows results. In this way, we offer the analyst the possibility to focus on control flow rather than non-triggering flows during the analysis phase (common criticism against conventional data flow approaches [19]) and thus, we achieve a strict formal separation between data and control in the system. Figure 7.18 shows the Library Case where only control flows are included.

![Diagram of PID specification](image)

Figure 7.18: PID specification where only control flows are shown.

Formally, a control flow filter $\mathcal{F}_{\text{control flow}}$ can be expressed as follows:
\[ \mathcal{F}_{\text{controlflow}}: V^*_{PID} \rightarrow V_{\text{controlflow}} \]

where

\[ V^*_{PID} = (\text{Processes, ERT_views, External_agents, Timers, Flows, Ports}) \]
\[ V_{\text{controlflow}} = (\text{Processes, ERT_views, External_agents, Timers, A (Flows), Ports}) \]
\[ A (\text{Flows}) = \{ F \mid F \in \text{Flows}, C (F) = \text{triggering_flow} \} \]

The function \( C (P) \) finds the type of an instance \( P \).

**ERT_view Filters**

An ERT_view filter removes ERT_views from a PID specification. The ERT_view filter can be formally defined as follows:

\[ \mathcal{F}_{\text{ERT-view}}: V^*_{PID} \rightarrow V_{\text{ERT-view}} \]

where

- \( V^*_{PID} = (\text{Processes, ERT_views, Flows^*_{PID}, External_agents, Timers, Ports}) \)
- \( V_{\text{ERT-view}} = (\text{Processes, } \emptyset, \text{Flows}_v, \text{External_agents, Timers, Ports}) \)
- \( \text{Flows}_v \subseteq \text{Flows}^*_{PID} \)
- \( \text{Flow} = (\text{Name, Flow, Origin, Destination}) \)
- \( \text{Flow}_v \) is a set of \( \text{Flow} \)
- \( (\text{Flow}_\text{name}, \text{Origin, Destination}) \in \text{Flows}_v \rightarrow \)
  \( (\text{Origin} \in \text{Processes} \lor \text{Origin} \in \text{External_agents} \lor \text{Origin} \in \text{Timers}) \land \)
  \( (\text{Destination} \in \text{Processes} \lor \text{Destination} \in \text{External_agents} \lor \text{Destination} \in \text{Timers}) \)

Figure 7.19 shows the PID specification of the acquisition activity in Figure 7.8 where the ERT_views are removed by performing the ERT_view filter.
Figure 7.19: PID specification where the ERT.views are removed.

External Agent Filters

An external agent filter removes external agents from a PID specification. The external agent filter can be formally defined as follows:

$$F_{external\_agent}: V_{PID}^* \rightarrow V_{External\_agent}$$

where

- $$V_{PID}^* = \langle Processes, ERT.views, Flows_{PID}, External\_agents, Timers, Ports \rangle$$
- $$V_{external\_agent} = \langle Processes, ERT.views, Flows_v, \emptyset, Timers, Ports \rangle$$
- $$Flows_v \subseteq Flows_{PID}$$
- $$Flow = \langle Name\_Flow, Origin, Destination \rangle$$
- $$Flows_v$$ is a set of Flow
(Flow\_name, Origin, Destination) ∈ Flows\_v → 
(Origin ∈ Processes ∨ Origin ∈ ERT\_views ∨ Origin ∈ Timers) ∧ 
(Destination ∈ Processes ∨ Destination ∈ ERT\_views ∨ Destination ∈ Timers)

**Timer Filters**

A timer filter removes timers from a PID specification. The timer filter can be formally defined as follows:

\[ F_{\text{timer}}: V_{PID}^* → V_{\text{timer}} \]

where

- \( V_{PID}^* = (\text{Processes}, \text{ERT\_views}, Flows_{PID}^*, \text{External\_agents}, \text{Timers}, \text{Ports}) \)
- \( V_{\text{timer}} = (\text{Processes}, \text{ERT\_views}, \text{Flows}_v, \text{External\_agents}, \emptyset, \text{Ports}) \)
- \( Flows_v ⊆ Flows_{PID}^* \)
- \( \text{Flow} = (\text{Name\_Flow}, \text{Origin}, \text{Destination}) \)
- \( \text{Flows}_v \) is a set of Flow

- (Flow\_name, Origin, Destination) ∈ Flows\_v → 
  (Origin ∈ Processes ∨ Origin ∈ ERT\_views ∨ Origin ∈ External\_agents) ∧ 
  (Destination ∈ Processes ∨ Destination ∈ ERT\_views ∨ 
   Destination ∈ External\_agents)

**Process Overview Filters**

A process overview hierarchy may be deduced from a PID specification including all levels of decomposition. For such a specification, all components except processes are abstracted away. Each process in the PID is only allowed to be represented by one icon in the overview. Processes are connected by links means that one process is in the decomposition of the other. Thus, each link denotes a hierarchical relationship between the processes it connects.

Figure 7.20 shows parts of the process hierarchy diagram generated from the PID specifications of the Library Case Study.

This filter differs from the filters defined above by the fact that it refers to several specification levels of a TEMPORA specification; the others refer only to one level
Figure 7.20: Process hierarchy.

at a time. We have however, decided to include it as an filter type since it is a simplification of PID and can be used much in the same way as the above mentioned filters.

Formally, a process overview filter $F_{overview}$ can be expressed as follows:

$$F_{overview}: V_{PID}^* \rightarrow H_{overview}$$

where

$$V_{PID}^* = \langle \text{Processes, ERT.views, ExternalAgents, Timers, Flows, Ports} \rangle$$

$$H_{overview} = \langle \text{Processes, } \mathcal{L} \text{ (Processes)} \rangle$$

$$\mathcal{L} \text{ (Processes)} = \{ (P_1, P_2) \mid \mathcal{P} (P_1, P_2) \}$$

where $\mathcal{P} (P_1, P_2)$ holds if process $P_1$ is the parent of process $P_2$ (i.e., $P_2$ is a process in the decomposition of $P_1$).

**PID Component Filters**

A particular process can be selected from a full specification for instance by clicking on the object in a GUI\(^4\), and appropriate information associated with this process is extracted and presented, including:

---

\(^4\)GUI is an acronym for graphical user interface.
• ports
• input flows from \{\text{sender}\}
• output flows to \{\text{receiver}\}

Each of the items above may be presented in different ways; Ports may be abstracted (Figure 7.21) or shown in full (Figure 7.22) and sender/receiver objects may be denoted by text (Figure 7.21) or shown with their graphical symbols (Figure 7.22). Optionally, one may also specify how many associated components that should be shown.

![Diagram](image)

Figure 7.21: Process P4.6 and associated information (PID symbols left out).

The formal definition of the PID component filter \( F_{\text{component}} \) is found in Appendix F.

### 7.4 ERT Filters

This section suggests a set of filters to generate various viewspecs of ERT specifications. The list of filters that we define for the ERT in this thesis is as follows:

• Language filters:
Figure 7.22: Process P4.6 and associated information.

- **Entity class filters** are inclusive and remove all components from a specification except for entity classes.

- **Value class filters** are exclusive and remove all value classes.

- **Timestamping filters** are either inclusive or exclusive. Inclusive timestamping filters keep time stamped entity classes and relationship classes but removes all other components, whereas exclusive timestamping filters remove the time stamping from entity classes and relationship classes.

- **Generalisation filters** are either inclusive or exclusive, thus, keep or remove generalisation hierarchies from a specification.

- **Complex entity class filters** is exclusive and remove all complex entity classes.

- **Derived information filters** is exclusive and remove all derived components.

- **Cardinality constraint filters** is exclusive and remove all cardinality constraints.

- **Model filters:**

  - **Component filters** are inclusive and keep a selected entity together with its immediate components.

To illustrate the various filters, the acquisition activity in the Library Case Study is used as example (Figure 7.23). Due to the space constraints we have left out the value classes in this diagram. Figure 7.24 shows a more detailed diagram of some of the components.
Figure 7.23: ERT specification of parts of the acquisition system.

Figure 7.24: A more detailed specification of the objects Client and Bibliographic document.
Entity Class Filters

The entity class filter removes all components from a specification except for entity classes. It can be formally defined as follows:

\[ F_{\text{entity\_class}}: V^*_E \rightarrow V_{\text{entity\_class}} \]

where

- \( V^*_E = \langle \text{Entity\_class}, \text{Relationship\_class}, \text{Is\_a\_relationship}, \text{Value\_class}, \text{Complex\_entity\_class}, \text{Cardinality\_constraint} \rangle \)
- \( V_{\text{entity\_class}} = \langle \text{Entity\_class}, \emptyset, \text{Is\_a\_relationship}, \emptyset, \text{Complex\_entity\_class}, \emptyset \rangle \)

Figure 7.25 shows the viewspec that results when an entity class filter is performed on the ERT diagram in Figure 7.23.

Figure 7.25: Only entity classes and their generalisation hierarchies are kept in the diagram.
Value Class Filters

A value class filter performed on an ERT specification removes all value classes. The value class filter can be formally defined as follows:

\[ \mathcal{F}_{\text{value\_class}}: V^*_\text{ERT} \rightarrow V_{\text{value\_class}} \]

where

- \( V^*_\text{ERT} = \langle \text{Entity\_class}, \text{Relationship\_class}^*_\text{ERT}, \text{Is\_a\_relationship}, \text{Value\_class}, \text{Complex\_entity\_class}, \text{Cardinality\_constraint}^*_\text{ERT} \rangle \)
- \( V_{\text{value\_class}} = \langle \text{Entity\_class}, \text{Relationship\_class}_v, \text{Is\_a\_relationship}, \emptyset, \text{Complex\_entity\_class}, \text{Cardinality\_constraint}_v \rangle \)
- \( \text{Relationship\_class}_v \subseteq \text{Relationship\_class}^*_\text{ERT} \)
- \( \text{Relationship\_class} = (\text{Name\_relationship\_class}, \text{Component1}, \text{Component2}) \)
- \( \text{Relationship\_class}_v \) is a set of \( \text{Relationship\_class} \)
- \( (\text{Name\_relationship\_class}, \text{Component1}, \text{Component2}) \in \text{Relationship\_class}_v \rightarrow (\text{Component1} \notin \text{Value\_class}) \land (\text{Component2} \notin \text{Value\_class}) \)
- \( \text{Cardinality\_constraint}_v \subseteq \text{Cardinality\_constraint}^*_\text{ERT} \) where each element of \( \text{Cardinality\_constraint}_v \) is associated with an element of \( \text{Relationship\_class}_v \)

Figure 7.26 shows the viewspec that results when a value class filter is performed on the ERT diagram in Figure 7.24.

Timestamping Filters

We distinguish between two types of timestamping filters: exclusive timestamping filters and inclusive timestamping filters.

An exclusive timestamping filter performed on an ERT specification removes the time stamping from entity classes and relationship classes. The exclusive timestamping filter can be formally defined as follows:

\[ \mathcal{F}_{\text{timestamping\_excl}}: V^*_\text{ERT} \rightarrow V_{\text{timestamping\_excl}} \]

where
7.4. ERT Filters

Figure 7.26: The value classes are removed from the diagram.

- $V_{ERT} = \langle Entity\_class, Relationship\_class, \text{Is\_a\_relationship, Value\_class, Complex\_entity\_class, Cardinality\_constraint} \rangle$

- $V_{timestamping\_excl} = \langle A \(Entity\_class\), B \(Relationship\_class\), \text{Is\_a\_relationship, Value\_class, Complex\_entity\_class, Cardinality\_constraint} \rangle$

- $A \(X\)$ is a function that converts all timestamped entity classes in $Entity\_class$ into entity classes and leaves the other elements of the set unchanged.

- $B \(X\)$ is a function that converts all timestamped relationship classes in $Relationship\_class$ into relationship classes and leaves the other elements of the set unchanged.

Figures 7.27 shows the viewspec that results when an exclusive time filter is performed on the ERT diagram in Figure 7.23.

An inclusive timestamping filter performed on an ERT specification keeps time stamped entity classes and relationship classes but removes all other components. Figure 7.28 shows the viewspec that results when an inclusive time filter is performed on the ERT diagram in Figure 7.23.

The formal definition of the inclusive timestamping filter $\mathcal{F}_{timestamping\_incl}$ is found in Appendix F.

**Generalisation Filters**

**Inclusive generalisation filter** An entity is chosen and all its subtypes and supertypes are shown. The inclusive generalisation filter can be formally defined as
Figure 7.27: The timestamped information is removed from the diagram.

follows:

$$\mathcal{F}_{generalisation_{incl}}: V_{ERT}^* \longrightarrow V_{generalisation_{incl}}$$

where

- $$V_{ERT}^* = (Entity\_class^{ERT}, Relationship\_class, Is\_a\_relationship^{ERT}, Value\_class, Complex\_entity\_class, Cardinality\_constraint)$$
- $$V_{generalisation_{incl}} = (A (Entity\_class^{ERT}), \emptyset, B (Is\_a\_relationship^{ERT}), \emptyset, \emptyset)$$
- $$is\_a^* (X,Y) = is\_a (X,Y) \lor (is\_a^* (X,Z) \land is\_a^* (Z,Y))$$
- $$\text{supertype} (X) = \{ Y | is\_a^* (X,Y) \}$$
- $$\text{subtype} (X) = \{ Y | is\_a^* (Y,X) \}$$
- $$A (E) = \{ X \} \cup X \in \text{subtypes} (X) \cup \text{supertypes} (X))$$
- $$Is\_a\_relationship_v \subseteq Is\_a\_relationship^{ERT}$$
7.4. ERT Filters

Figure 7.28: Only the timestamped information is kept in the diagram.

- \( \text{Is\_a\_relationship} = \langle \text{Component1, Component2} \rangle \)
- \( \text{Is\_a\_relationship}_v \) is a set of \( \text{Is\_a\_relationship} \)
- \( B (\text{Is\_a\_relationship}_v^{ERT}) = \{ X \mid X \in \text{Is\_a\_relationship}_v^{ERT} \land \) 
  \( \langle \text{Component1, Component2} \rangle \land \) 
  \( \text{Component1} \in A (\text{Entity\_class}_v^{ERT}) \land \) 
  \( \text{Component2} \in A (\text{Entity\_class}_v^{ERT}) \) \)

Figure 7.29 shows the viewspec that results when an inclusive generalisation filter is performed on the ERT diagram in Figure 7.23.

**Exclusive generalisation filter** All its is_a relationships are removed. The exclusive generalisation filter can be formally defined as follows:

\[
\mathcal{F}_{\text{generalisation\_excl}}: V_v^{ERT} \rightarrow V_{\text{generalisation\_excl}}
\]

where
Figure 7.29: Only entity classes and their generalisation hierarchies are kept in the diagram.

- $V^*_{ERT} = \langle \text{Entity\_class}, \text{Relationship\_class}, \text{Is\_a\_relationship}_{ERT}, \text{Value\_class}, \text{Complex\_entity\_class}, \text{Cardinality\_constraint} \rangle$
- $V_{generalisation\_ excl} = \langle \text{Entity\_class}, \text{Relationship\_class}, \emptyset, \text{Value\_class}, \text{Complex\_entity\_class}, \text{Cardinality\_constraint} \rangle$

Figure 7.30 shows the viewspec that results when an exclusive generalisation filter is performed on the ERT diagram in Figure 7.23.

Complex Entity Class Filters

The complex entity class filter performed on an ERT specification removes complex entity classes. The complex entity class filter can be formally defined as follows:

$$\mathcal{F}_{\text{complex\_entity\_class}}: V^*_{ERT} \rightarrow V_{\text{complex\_entity\_class}}$$

where

- $V^*_{ERT} = \langle \text{Entity\_class}, \text{Relationship\_class}_{ERT}, \text{Is\_a\_relationship}, \text{Value\_class}, \text{Complex\_entity\_class}, \text{Cardinality\_constraint}_{ERT} \rangle$
- $V_{\text{complex\_entity\_class}} = \langle \text{Entity\_class}, \text{Relationship\_class}_v, \text{Is\_a\_relationship}, \text{Value\_class}, \emptyset, \text{Cardinality\_constraint}_v \rangle$
Figure 7.30: All generalisation hierarchies are removed from the diagram.

- $\text{Relationship} \subseteq \text{Relationship}^{\text{ERT}}$

- $\text{Relationship} = \langle \text{Name} \_\text{relationship}, \text{Component}1, \text{Component}2 \rangle$

- $\text{Relationship}$ is a set of $\text{Relationship}$

- $\langle \text{Name} \_\text{relationship}, \text{Component}1, \text{Component}2 \rangle \in \text{Relationship}$

  $\rightarrow (\text{Component}1 \in \text{Entity} \_\text{class} \lor \text{Component}1 \in \text{Value} \_\text{class}) \land$

  $(\text{Component}2 \in \text{Entity} \_\text{class} \lor \text{Component}2 \in \text{Value} \_\text{class})$

- $\text{Cardinality} \subseteq \text{Cardinality}^{\text{ERT}}$, where each element of $\text{Cardinality}$ is associated with an element of $\text{Relationship}$

Figure 7.31 shows the viewspec that results when a complex entity class filter is performed on the ERT diagram in Figure 7.26.

**Derived Information Filters**

The derived information filter performed on an ERT specification removes all derived components, i.e., derived entity classes, derived value classes, and derived relation-
Figure 7.31: The complex entity and value classes are removed from the diagram.

ship classes are removed. The derived information filter can be formally defined as follows:

\[ F_{\text{derived\_information}} \colon V_{\text{ERT}}^* \rightarrow V_{\text{derived\_information}} \]

where

- \( V_{\text{derived\_information}} = \langle A(\text{Entity\_class}), B(\text{Relationship\_class}), Is\_a\_relationship, C(\text{Value\_class}), \text{Complex\_entity\_class}, \text{Cardinality\_constraint} \rangle \)

- \( A(\text{Entity\_class}) = \{ X \mid X \in \text{Entity\_class}, D(X) \neq \text{derived\_entity} \} \)

The function \( D(P) \) finds the type of an instance \( P \).

- \( B(X) = \{ X \mid X \in \text{Relationship\_class}, D(X) \neq \text{derived\_relationship\_class} \} \)

- \( C(X) = \{ X \mid (X \in \text{Value\_class}, D(X) \neq \text{derived\_value\_class}) \} \)

- \( \text{Relationship\_class}_v \subseteq \text{Relationship\_class}^*_\text{ERT} \)

- \( \text{Relationship\_class} = \langle \text{Name\_relationship\_class}, \text{Component1}, \text{Component2} \rangle \)

- \( \text{Relationship\_class}_v \) is a set of \( \text{Relationship\_class} \)
- \((\text{Name\_relationship\_class}, \text{Component1}, \text{Component2}) \in \text{Relationship\_class}_v \rightarrow (\text{Component1} \in A (\text{Entity\_class}) \lor \text{Component1} \in C (\text{Value\_class})) \land (\text{Component2} \in A (\text{Entity\_class}) \lor \text{Component2} \in C (\text{Value\_class}))\)

- \(\text{Cardinality\_constraint}_v \subseteq \text{Cardinality\_constraint}_{ERT}\) where each element of \(\text{Cardinality\_constraint}_v\) is associated with an element of \(\text{Relationship\_class}_v\).

Figure 7.32 shows the viewspec that results when a derived information filter is performed on the ERT diagram in Figure 7.23.

**Cardinality Constraint Filters**

A cardinality constraint filter performed on an ERT specification removes all cardinality constraints. It can be formally defined as follows:

\[ F_{\text{cardinality\_constraint}}: V^*_v \rightarrow V_{\text{cardinality\_constraint}} \]

where
- $V_{\text{ERT}} = \{\text{Entity\_class}, \text{Relationship\_class}_{\text{ERT}}, \text{Is\_a\_relationship}, \text{Value\_class}, \text{Complex\_entity\_class}, \text{Cardinality\_constraint}_{\text{ERT}}\}$
- $V_{\text{cardinality\_constraint}} = \{\text{Entity\_class}, \text{Relationship\_class}_{\text{v}}, \text{Is\_a\_relationship}, \text{Value\_class}, \text{Complex\_entity\_class}, \emptyset\}$

Figure 7.33 shows the viewspec that results when a value class filter is performed on the ERT diagram in Figure 7.24.

![ERT Diagram]

**Figure 7.33:** The cardinality constraints are removed from the diagram.

### Component Filters

A particular entity can be selected from a full specification for instance by clicking on the object in a GUI, and appropriate information associated with this entity is extracted and presented. Figure 7.34 shows the viewspec that results when an ERT component filter is performed on the ERT diagram in Figure 7.24 and the entity Client is chosen.

The formal definition of the ERT component filter $\mathcal{F}_{\text{component}}$ is found in Appendix F.

### 7.5 ERL Filters

This section suggests a set of filters to generate various viewspecs of ERL specifications.
We do not suggest filters for the textual representation of rules but rather concentrate on defining filters for focusing on relevant relationships between rules, and relevant relationships between rules and other TEMPORA models. The rationale behind this choice is that the problem of manipulation of single rules as reported in, e.g., [142, 143], is due to the lack of a syntax-oriented rule editor\textsuperscript{5} and the lack of systematical ways for splitting a rule into simpler rules. In the current version of ERL, the size of a rule can be kept at a reasonable level through the use of predicates which allow a complex rule to be split into simpler rules (see below).

From the above, we conclude that the notion of filters that only operate on the textual parts are superfluous. However, case studies have shown that we need to provide support for how components of a rule relates to other rules as well as other TEMPORA models without being disturbed by irrelevant details. How the inter-language links can be used to support the creation of filters, e.g., to exploit the diagrammatic representation of the other TEMPORA languages, is postponed to the next chapter.

### Defining Viewspecs Through Rule Relations

A rule base may be viewed from different perspectives through the set of rule relations which are defined in the previous chapter. The relations relate rules in the rule base and each relation type corresponds to a specific perspective. The rule filter can be defined as follows:

\[
\mathcal{F}_{\text{relation}} : V^{2}_{\text{ERL}} \rightarrow V_{\text{relation}}
\]

\textsuperscript{5}A syntax-oriented rule editor was not implemented in TEMPORA due to resource limits. How such an editor may mend some of the problems of manipulation of single rules are dealt with in Chapter [142, 143].
where

- \textit{relation} is either of the following: \texttt{causes}, \texttt{uses}, \texttt{has\_info\_derived\_by}, \texttt{constrained\_by}, \texttt{relates\_to}, \texttt{suspends}, and \texttt{overrules}.
- \( V_{ERL} = (\langle R \rangle, \langle R \rangle) = \{ (\alpha, \beta, \delta, \tau) \} \)
- \( V_{relation} = (\langle R' \rangle, \langle R' \rangle) = \{ (\alpha', \beta', \delta', \tau) \} \)
- \( V_{ERL} \subseteq V_{ERL}^* \)

For example, all event action rules in the library system can be found by selecting the \texttt{causes} relation, i.e., \( F_{relation} \) where \texttt{relation} = \texttt{causes}. However, in most cases one is not interested in all the rules of a particular type. Figure 7.35 shows a subset of rules related by the \texttt{relates\_to} relation. In this case, \texttt{relates\_to} denotes the \texttt{motivates} and \texttt{necessitates} relations. The relations are used to describe hierarchies of purpose (correspond to goal hierarchies in organization theory). The \texttt{motivate} relation denotes the fact that the existence of one rule is a consequence of the fact that the other holds. The \texttt{necessitates} relation expresses an even stronger relationship between two rules. It indicates that rule A is at a higher level than rule B and the contents of B is necessary to achieve the contents of A.

![Figure 7.35: An example hierarchy of purpose: the motivates and necessitates relations](image)

Subsets of each relation type can be expressed formally as follows:

\[ F_{Rel} (X): V_{ERL}^* \rightarrow V_{Rel} \]

where
7.6 Chapter Summary

- \( \text{Rel} \in \{ \text{causes, uses, has\_info\_derived\_by, constrained\_by, relates\_to, suspends, overrules} \} \)

- \( \text{V}_{\text{Rel}} = \langle R \rangle \),
  \[
  R = \{ Y \mid X = Y \lor \text{Rel}^* (X,Y) \lor \text{Rel}^* (Y,X) \}
  \]

- \( \text{Rel}^* \) is a transitive closure of any combination of the relations.

This chapter has presented a framework for filtering mechanisms. We have defined basic concepts and suggested ways of characterising abstraction levels, views specs and filters. Information may be expressed at different levels of detail and we refer to a specific level of detail as an abstraction level. The notion of abstraction level say something about what we want to focus on in the domain/system and at what level of detail we want to examine the relevant aspects of the domain. Which abstraction level is desirable may vary during the development process. For example, at one point we want to look at all details of a PID specification and at a different point only a subset of the same specification may be sufficient. Enforcing the user to use the same specification in both cases means that the user may be disturbed by a large number of irrelevant details. To deal with the generation of specifications at various abstraction levels, we have introduced the concepts of filters and views specs. A filter is defined as an abstraction of a specification and the resulting simplified specification is referred to as a viewspec.

Views specs are divided into three types, where each may differ with respect to what languages that are used to represent its subviews specs and what specification levels these are located at: intra-language views specs (same language and same specification level), inter-language views specs (different languages and same specification level) and inter-level viewspec (same or different languages and different specification levels). The use of different types of views specs is elaborated in the next chapter. Filters have been classified as language (or meta-model) and model (specification) filters. The former is used to suppress details in a specification with respect to constructs of the language that is used for its representation, whereas the latter is defined with respect to a particular model or specification. Moreover, we have defined a formal basis for defining different types of filters for a specific language. This is used as a basis for formally defining a set of filters for the TEMPORA languages.

To generate views specs of TEMPORA specifications, we have described the following set of filters: PID filters, ERT filters and ERL filters. The PID and ERT filters provide the developer with facilities to abstract away complexity reducing constructs from a specification as well as other irrelevant details. The ERL filters do not operate on a single rule but they rather provide means to view a collection of rules. Thus, the ERL filters can be used to focusing on relevant relationships between rules by using the rule relations defined in the previous chapter or a subset of these. The
result of performing any of the TEMPORA filter is an intra-language viewspec. In
the next chapter, we will show how the filtering mechanisms can be used to present
subsets of a specification database and as a basis for entering more information into
the specification database.
Chapter 8

The Complexity Reduction Approach

Some examples of how the complexity reduction approach can be used in the modelling process were provided in Chapter 1. It showed one way of adding details to a DFD diagram based on viewspecs. Viewspecs were also used to assist the validation of the diagram. In this chapter, we provide a more comprehensive and systematic presentation of the approach. It addresses how the structuring and filtering mechanisms defined in Chapters 5 to 7 can be used in the development process. It also examines the process of building up specifications based on viewspecs and proposes mechanisms to support it.

Section 8.1 provides an overview of the main activities of the complexity reduction approach. The Sections 8.2 to 8.5 detail each of the activities. Section 8.6 outlines how the suggested approach relates to the TEMPORA development process.

8.1 An Overview of the Approach

The approach based on systematic complexity reduction is divided into four major activities as shown in Figure 8.1 (the activities are not necessarily sequential):

- Filtering is the process of generating viewspecs from a specification, i.e., creating appropriate filters.

- Presentation is the process of using the viewspecs that result from the filtering in a way that facilitates communication and understanding.

- Entering new specification details is the process of using viewspecs as a basis for entering new specification details, corresponding to the updating of viewspecs.
- *Inclusion of changes* is the process of building a revised version of a full specification based on updated viewspecs.

![Diagram](image)

**Figure 8.1: The approach based on systematic complexity reduction.**

Figure 8.1 is a revision of Figure 1.3 and provides a more detailed picture of the complexity reduction approach. It shows that *several* viewspecs can be generated from a full specification, that viewspecs can be generated from other viewspecs and that *more than one* viewspec can be updated and used as a basis for *revising* the full specification.

**An Example**

To illustrate central issues of each of the above activities, we use the Sweden Post Case Study. In Chapter 1, a general description of the ordering and invoicing system was provided together with a top-level DFD (Figure 1.4). In this chapter, we provide a more detailed presentation which includes PID, ERT and ERL modelling. The PID model which shows all the main processes in the system is shown in Figure 8.2. The ERT model in Figure 8.3 shows an overview of the main concepts in the system. Appendix E provides further documentation (ERT diagram and rules). The ERT
model is documented in a set of ERT diagrams and the rules describe behaviour of the system, define constraints and derive information. An example of a rule is shown in Figure E.2 in Appendix E. The rule form implemented in the TEMPORA capture tool provides both a textual and a formal representation of a rule together with additional information such as name of developer, date of specification and connections to other parts of the TEMPORA model. The documentation of the Sweden Post case study is found in [100, 133, 136].

Figure 8.2: PID specification of the Sweden Post ordering and invoicing system.

As far as the modelling of the Sweden Post case study is concerned it is a result of a cooperative effort in the TEMPORA project. In particular, I would like to mention that Peter McBrien\textsuperscript{1} and Rolf Wohed\textsuperscript{2} have been heavily involved in the work. Although the author was responsible for the part described below we have been

\textsuperscript{1}Currently at King's College London, Strand, London.
\textsuperscript{2}SISU, Stockholm.
all working on the various specifications in an iterative manner, and it is therefore difficult to distinguish who has contributed what. The work on complexity reduction is however, entirely done by the author.

It is beyond the scope of this text to deal with all aspects of the system. We limit ourselves to elaborate on P6 Register delivery note in Figure 8.2. P6 Register delivery note consists of three main subactivities which are shown in Figure 8.4. The processes P6.1 Entering and P6.2 Checking are again decomposed (Figures 8.5 and 8.6). In the remainder of the chapter, we will look at the details of the PID, the ERT and the ERT specifications of the system as we need them for illuminating the major features of the complexity reduction approach.

It should be noted that the figures that illustrate the Sweden Post case study are not screen dumps of the TEMPORA capture tool (exceptions are the diagrams in Appendix E). Since only some complexity reducing features are implemented in the tool, the figures in the sequel are handmade. This also improves the perceptibility of the examples since we can manipulate and annotate the figures in a flexible manner. However, the tool has been very useful in generating the example and in examining different parts of the order and invoicing system. As we will see below the searching and navigation facilities are crucial in this work.
Figure 8.4: The decomposition of P6 Register delivery note.

Figure 8.5: PID specification of P6.1.
8.2 Filtering

This section examines how the features of filters influence the filtering process and also suggests different ways of supporting this process.

To define a filter implies that the user must determine the right abstraction level for the information in a specification which is necessary to carry out a task. Thus, the filtering process consists of two subactivities [115]:

1. Determine which specifications to focus on.
2. Determine which components of the selected specifications to focus on.

The first step means that the user must locate which specifications should form the basis for the filtering, i.e., one or more full specifications or one or more viewspecs. In the second step, the selected specifications from the first step are used as input to a filter. The result is a viewspec which contains the relevant details whilst the irrelevant details are suppressed. To what extent this process can be supported depends on the nature of the filter. The user of a filter may be a human being or another system, e.g., an explanation facility. An explanation generation facility invokes a filter in order to use the resulting viewspec as a part of the generated explanation. Thus, graphical viewspecs are used to together with textual explanations to enhance validation. The combination of execution techniques [153], complexity
8.2. Filtering

reducing techniques and presentation techniques are elaborated in [154]. Chapter 9
gives a brief description of how filters can be used together with the explanation
generation facility in the PPP tool environment.

Level of Automation

We envisage at least three ways of generating a viewspec:

- **Automatically.** The appropriate filter is invoked together with the necessary
  information. The viewspec is then created automatically by the filter. A
  viewspec generated by an automated filter can be derived from the originating
  specification at any time using the respective filtering operator. In certain
  cases it may however be useful to store the viewspec. For example, if a non-
deterministic filter is applied to a large specification it may be desirable to
  store the resulting viewspec to keep a familiar layout.

- **Semi-automatically.** The generation of a viewspec is partly supported by a
  filter, but manual intervention is necessary. A typical situation is found in
  the filtering process for diagrammatic languages where complex drawing must
  often be performed manually.

- **Manually.** The generation of a viewspec is entirely done by the user with-
  out any particular computerised support beyond general editing facilities. A
  viewspec may be produced manually for one of the two reasons: no filter can
  be defined to generate the desired viewspec or it is awkward to do so.

The extent to which automated support can be provided to generate an appropriate
viewspec using a particular filter is dependent on factors such as:

- Whether the filter can be formally expressed. If a filter is formally defined it
  has a higher potential for comprehensive computerised support than informal
  filters.

- Whether necessary information is available before filter invocation. Filters
differ with respect to what parts of a specification they operate on, i.e., some
  filters operate on the whole specification whereas others require an explicit
  selection of relevant parts. This may be done interactively by selecting relevant
  components (e.g., by clicking on the relevant object in a GUI), setting default
  values or by storing the necessary information in a file.

- Whether the output of a filter is deterministic or non-deterministic. The user
  may want to apply the filter only to certain parts of a specification at a time
  and she might decide what parts during the filtering process.
• Whether the filter has global or local effects. If the scope of effects of a filter is global, manual interventions might be necessary to ensure a controlled propagation of changes to affected specifications.

• Whether the filter requires heavy computation. For instance, if the user works interactively, reasonable response times are required. Computations which take hours and days are out of the question.

Pre-Defined versus User-Defined Filters

We distinguish between two major groups of filters: pre-defined filters and user-defined filters. Pre-defined filters for the TEMPORA languages include all filters that are well-defined, i.e., the filters described in the previous chapter. User-defined filters are all other filters. Table 8.1 provides an overview of relevant aspects of pre-defined and user-defined filters. A user-defined filter is manually produced and is characterised by its non-deterministic property. It is up to the developer to decide what details that should be highlighted and what details that should be suppressed. We may envisage different ambition levels as far as user-defined filters are concerned. In the simplest case, no control is provided by the system and support for creating such filters is limited, e.g., to drawing support for a visual language. We may also envisage more advanced environments which could offer the developer facilities to define new filters which could be integrated into the modelling environment in a similar manner as the pre-defined filters. However, before such facilities can be developed we need more experience using the filtering mechanisms in field studies (see Chapter 10).

In contrast to user-defined filters, pre-defined filters lend themselves to comprehensive support because of their formal properties. Pre-defined filters are therefore preferred to user-defined filters as far as automated support is concerned. Pre-defined filters also open up for an interesting feature, namely that filters can be used in an active manner. By this we mean that some ‘intelligence’ can be built into a tool to provide the user with appropriate viewspecs according to what she is doing, e.g., based on user modelling. For instance, a developer can tailor the use of filters to fit better with the tasks to be undertaken by providing a set of default values as input to automated filters that are frequently used. In this way, the user does not need to provide the same information as input each time a filter is used.

To exploit the potential of filtering mechanisms, it is therefore important to identify a sufficient set of pre-defined filters, i.e., filters which are frequently used or time-consuming to create. Filters may be defined with respect to a particular language as well as to a specific system/domain. Typically, one would pre-define filters for all complexity increasing construct of a language, e.g., flows and ports in the PID. The modelling process is a creative activity and therefore, it will be impossible to pre-define all kinds of filter which a user may need. The user should always have the possibility to define her own filters.
### Composed Filters

Filters can be concatenated to form *composed filters*. Performing a composed filter means that two or more *simple filters*\(^3\) are performed in consecutive order. A composed filter consists of two or more pre-defined filters or combinations of pre-defined and user-defined filters. An example of a composed PID filter is shown in Figure 8.7. It consists of two pre-defined filters, where an inclusive component filter is followed by an exclusive ERT view filter. The inclusive component filter is performed on the PID specification in Figure 8.2 where P6 Register delivery note is selected. The two filters are nested in the sense that the output from the first filter is used directly as input to the next filter. Both the intermediate and resulting viewspecs generated by this filter are included in Figure 8.7.

To allow nesting of filters which are performed automatically requires that:

1. The filters operate on the same type of specification, i.e., the specifications are expressed using the same language.
2. The format of the filters are the same, c.f., definition of a filter in Section 7.1. Only if this is true, the output from one filter can be used as input to another filter.

The sequence of performing the filters that make up a composite filter may be crucial for the result, i.e., a composed filter is not necessarily symmetric. We must therefore record the sequence of the subfilters if we want to reproduce a viewspec produced by a composed filter.

---

\(^3\)A filter with a single purpose is referred to as simple. All the filters defined in the previous chapter are simple.
Figure 8.7: An example of a composite filter.

Filtering Across Specification Boundaries

So far we have mostly used the pre-defined filters for the TEMPORA languages to generate intra-language viewspecs which:

- are projections or approximations of the originating specifications,
- are represented in the same language as the originating specifications and
- are located at the same level of specification as the originating specifications.

The only exception is the rule filters which have been used to generate viewspecs that contain rules belonging to different specification levels, i.e., inter-level viewspecs. However, also these filters have been defined with respect to one type of specification and their output is the same type as the originating specification. In the remainder of this section, we will look at how we can carry out filtering across specification boundaries. As we will see below, this is necessary in order to generate inter-level viewspecs for specifications which are represented in the ERT and PID and to generate inter-language viewspecs. Filtering across specification boundaries is also necessary for generating intra-language viewspecs based on more than one specifications. Figure 8.8 shows an example of how we can define viewspecs in the ERT and PID. To summarise, a filter can be defined with respect to the following specifications:

**Type 1** One specification which can either be a full specification or a viewspec of that specification. In Figure 8.8, the resulting viewspecs are denoted $A$ and
8.2. Filtering

$B$, respectively. This correspond to the filtering dealt with so far in the thesis. To generate a viewspec ($B$) from another viewspec corresponds to a composed filter.

**Type 2** Two or more specifications located at the same specification level and represented in the *same* language (the resulting viewspec is denoted $C$ in Figure 8.8). This is possible by utilising intra-language links.

**Type 3** Two or more specifications expressed using *different* languages (the resulting viewspec is denoted $D$ in Figure 8.8). This is possible by utilising inter-language links.

**Type 4** Two or more specifications located at different specification levels but represented in the *same* language (the resulting viewspec is denoted $E$ in Figure 8.8). This is possible by utilising inter-rule links and inter-level links.

![Diagram](image)

**Figure 8.8**: An overview of viewspec generation.

From the above, we can envisage a wide range of specific filters that can be defined with respect to the four types of originating specifications. For instance, filters may be defined in such a way that the same viewspec is produced from different originating specifications (Figure 8.9) and vice versa (Figure 8.10). The resulting viewspecs in the examples provided in Figures 8.9 and 8.10 are all intra-language viewspecs. The result could as well have been an inter-language or an inter-level viewspec. It is a *user-decision* to determine what viewspecs that should be generated. The possibility of creating filters across specification boundaries implies that the number of filters that can be defined are *infinite*. Rather than defining a set of filters to generate a limited set of viewspecs, we advocate an approach to filtering which utilises the existing pre-defined filters in combination with the structuring mechanisms defined in Chapters 5 and 6. As we will show by means of examples below, this facilitates definition of filters across specification boundaries in a flexible manner.
Figure 8.9: Different filters may be used to generate the same viewspec.

Figure 8.10: The same specifications may be used to generate different viewspecs.
How Structuring Facilitates the Filtering Process

Besides being essential in the generation of viewspecs using the pre-defined filters, the links facilitate flexible filtering across specification boundaries by providing the basis for *advanced searching and navigation within and across specifications*. Filtering across specification boundaries is done by performing the following steps one or more times:

**Step 1** Searching and navigation functions are used to find the specifications that contain the relevant components. The links facilitate easy access from one specification to related components contained in other specifications, and are used as a basis for providing the user with overviews of relevant components.

**Step 2** For each element in the set of specifications that result from step 1, the pre-defined filters are used to generate desired intra-language viewspecs (located at the same or different specification levels). This corresponds to simple and composite filters that take one specification as input and then generate a viewspec represented in the same language.

Figure 8.11 shows an overview of the steps of the filtering when performed across specification boundaries. The details of the diagram is described in Example 1 below.

A filter may also work upon more than one specification, e.g., by finding immediately connected components of an entity class in different ERT diagrams (see below). The result of performing steps 1 and 2 may result in one or more viewspecs. As we will see in the next section, the intra-language viewspecs may then be put together to form composite viewspecs (inter-language and inter-level viewspecs). We will now illustrate with some examples how filtering across specification boundaries can be performed.

**Example 1: Finding ERT objects associated with a PID process** We choose to look at the process which describes the registration of delivery notes, more specifically the process *P6.1.2 Enter delivery note head*. A viewspec that shows this process and its immediate components is shown in Figure 8.12. It is generated by means of an inclusive PID component filter performed on the specification in Figure 8.5 where process *P6.1.2 Enter delivery note head* is selected. We then want to find all ERT objects that are associated with this process and then examine different aspects of some of these objects. This can be done by using the viewspec in Figure 8.12 as a basis, activating a search or navigation function and then selecting process *P6.1.2 Enter delivery note head* as input.

If a search function is invoked, the user will typically see a list which contains the relevant ERT objects associated with the process, e.g., in a separate window on the screen (Figure 8.13a). If a navigation function is invoked, the user may choose if she wants to see the actual ERT diagrams that contain the relevant objects in addition
Figure 8.11: Filtering across specification boundaries.

Figure 8.12: Viewspec that shows P6.1.2 Enter Delivery Note Head and its immediately connected components.
to the list of relevant objects (Figure 8.13b). The resulting objects may be found in more than one diagram. If this is the case the different diagrams may be shown, e.g., in separate windows or the user may choose which diagram to be shown at a time. Such diagrams can also be used as a basis for further searching, navigation or modelling.

![Diagram](image)

**Figure 8.13:** The relevant objects associated with P6.1.2 Enter Delivery Note Head.

As we see in Figure 8.13b, we find the relevant objects associated with P6.1.2 Enter delivery note head in several diagrams. We will limit ourselves to show how we can examine some aspects of the objects: Delivery_note and Agreement. To get an overview of how they relate to other major concepts of the system, we use the diagram in Figure 8.3 as a basis. The Figures 8.14 and 8.15 show the major features of Delivery_note and Agreement, respectively. The viewspecs are generated by performing an inclusive ERT component filter for the different objects, e.g., choose Delivery_note as input and specify that only immediately connected components should be included in the viewspec, i.e., \( N = 1 \). An alternative component filter is to extract all immediately connected components irrespective of which specifications these are located in. Moreover, the objects' classification can be examined by choosing an inclusive generalisation filter (Figure 8.16). In a similar manner other aspects of these objects could have been examined by performing any of the ERT filters defined in Chapter 7.

As we see from the examples above, we define all filters with respect to an existing specification and in addition, we use search functions when the desired information is to be found across specification boundaries. We could however have chosen one object, generated a new diagram which contained all its connections and immediate components, and then used the filters on this diagram. However, in such an approach we trade familiarity with automation, or we rely on rather advanced drawing algorithms, e.g., algorithms which allow constraints to be specified (see next section). A second alternative would be to adopt an approach similar to the ARIES approach [63] which has a common internal representation language to which all external languages are translated (see Chapters 2 and 10).
Figure 8.14: Delivery_note and its immediately connected subcomponents.

Figure 8.15: Agreement and its immediate subcomponents.

Figure 8.16: Generalisation hierarchies for Customer and Agreement.
Example 2: Exploiting the visual languages to find rules  It is also possible to navigate between specification components by using diagrammatic as well as non-diagrammatic components. Thus, by exploiting the coupling between the various languages, the lack of graphical notation and structuring features of the rule language can be compensated for. In TEMPORA, this is essential in managing a large number of rules in a rule base. For instance, we may use the graphical representation of the ERT and the PID to select certain parts of a specification and derive the relevant associated rules. For instance, the event-action rules associated with the different processes in P6 Register delivery note (Figure 8.4) can be found by using the implements link between PID and ERL, cf., Chapter 6. An overview of the resulting rules is shown in Figure 8.17.

Figure 8.17: Overview of the event-action rules associated with process P6.1, P6.2 and P6.3.

We can then examine the event action rules associated with, e.g., the process P6.1.2 Enter delivery note head, by activating a search function and selecting the process. The result is shown in Figure 8.18a. In this way, the PID are used to define viewspecs of a rule base. Similarly, the ERT can be used to define viewspecs of a rule base. Figure 8.18b shows an overview of rules associated with the Customer entity class.

Example 3: Examine the rule base through rule relations  The rule relations can also be used to examine the rule base. Figure 8.19 shows the caused-by relations for the event-action rules associated with P6 Register delivery note. Figures 8.20a and 8.20b show the has_information-derived-by and constrained_by relations for rule R206224 Check delivery note line - 4 (see Appendix E), respectively.
Figure 8.18: a) shows the event-action rules associated with P6.1.2 Enter Delivery Note Head and b) shows the rules associated with the entity class Customer.

Figure 8.19: caused-by relations, filled circles denote communication with timer, ERT_views or external agents.

Thus, a rule base can be viewed from different perspectives by using: rule relations, the PID, the ERT or a combination of these. To what extent filtering across specification boundaries can be automated depends on a number of factors such as the flexibility of the structuring mechanisms and how the input to a filter can be specified (see next chapter). How viewspecs that result from the filtering can be presented in an appropriate manner, is elaborated in the next section. This includes, e.g., how inter-language viewspecs can be built from a set of intra-language viewspecs.
8.3 Presentation

Presentation is the process of using the viewspecs in a way that:

- Facilitates communication and understanding between developers and end-users, e.g., to support elicitation and validation.
- Facilitates communication and understanding among developers, e.g., to provide relevant information which is used as a basis for modelling decisions.
- Eases the modelling process for a single developer, e.g., rule formulation.

Although filtering and presentation are intertwined processes we have chosen to deal with them separately. Filtering deals with what details to be included (contents) in a viewspec, whereas presentation deals with how these details are to be presented (layout) and used in different facets of modelling.

This section suggests how presentation can be supported and it also examines different ways of presenting the viewspecs to different actors involved in the modelling process. As we will see below, the use of viewspecs in modelling implies that we must deal with several different viewspecs at a time. These may also be of different types. Thus, the section also addresses how multiple viewspecs can be allowed to co-exist and how these can be used effectively in the modelling process.

Improving the Layout of Viewspecs

In the previous chapter, we combined filtering and layout modification without dealing with that explicitly. Filtering away certain details from a diagrammatic speci-
fication simplifies the diagram, and often restructuring of the diagram is desirable, e.g., locations of symbols make the diagram too sparse or lines are unnecessarily crossing. Thus, in most of the cases, filtering and layout modification were combined because of relaxed space constraints when information was filtered away from a specification. The layout of a specification can be changed by redrawing the components of a diagrammatic specification or by rewriting the components of a non-diagrammatic specification. The purpose is to produce a simpler and more comprehensible arrangement of the details of a specification without changing the semantics. Although we expect the restructuring of diagrams to be done manually, there exists work that may be utilized to provide some support. When components are removed, modification of layout can be supported by utilizing algorithms for graph drawing, e.g., minimising the number of crossing lines. In addition, rather than removing information from a specification, irrelevant information may be suppressed in other ways.

**Automatic layout algorithms** Much research has focused on development of automatic layout algorithms which are used to produce 'nice' drawing of graphs on a screen without user intervention. A distinction is made between those which allow layout constraints to be specified and those which do not.

The latter type of algorithms is only capable of producing an aesthetically pleasing drawing of a graph without taking into account user or application specific constraints [15]. They also seldom make use of information in the current layout when calculating the new layout [15]. The consequence of these deficiencies is that whenever a new layout is generated, only a minor change in the diagram may imply a major change to the diagram in the new layout. Thus, the user's familiarity with the diagram is lost whenever a new diagram is created. A variety of algorithms for improving the layout of large graphs are developed. Algorithms which support restructuring of hierarchical graphs to reduce the number of crossing lines are found in, e.g., [128]. Other algorithms are developed to eliminate lines in a graph by, e.g., by adding a new node and collapsing arcs which have the same starting and ending point [88].

More advanced algorithms allow user-specified layout constraints to be added to their descriptions together with constraints used in the current layout, e.g., the constraint based approach reported in [15]. An example of a constraint for a PID diagram would be to place external agents at the edges of the diagram. The constraints ensure a more stable layout of a graph throughout different versions of the layout. However, constraint based algorithms often suffer from bad run-time performance and their application for interactive use is therefore limited unless special optimisation, tuning, etc. are provided.
8.3. Presentation

Alternative ways of suppressing information So far we have only considered how irrelevant details can be removed during the filtering process whereas the relevant details are the only components that remain in a viewspec. There are also other ways of highlighting and suppressing details of a specification such as by highlighting relevant components by using, e.g., colours and different size of symbols, by shading of components, by using black boxes and distinguish visibility of layers, e.g., the approaches taken by.

Different Ways of Applying Viewspecs

The viewspecs that result from the filtering process can be applied in a variety of ways depending on the purpose of the presentation and the role, background, skills, etc. of the actors involved. The presentation may differ with respect to what types of viewspecs that are used (intra-language, inter-language and inter-level viewspecs) and what type of medium that is used for presentation (screen or paper).

Types of viewspecs used for presentation We will now show with examples how different types of viewspecs can be used in the modelling process.

Example 1: Elicitation and validation We have already illustrated in Chapter 1 how viewspecs can be used as a means to support elicitation and validation. In a similar manner, all the viewspecs that are generated in the previous section (e.g., Figures 8.12 and 8.14) can be used as a basis for validating different parts of ERL, ERT and PID specifications or they may be used for eliciting more information. The essential point is that the communication process with the end-users (as well as with other developers) is supported by reducing the amount of detail contained in specifications. For more details about the use of viewspecs as a basis for validation, we refer the interested reader to [68]. It describes a student project which was carried out to test the use of viewspecs in the modelling process (see also Chapter 10).

Example 2: Conceptualisation So far we have stated that structuring is a pre-requisite for flexible filtering, particularly for creating filters based on more than one specifications. The links can also be utilised after a viewspec is created. This holds particularly for languages that are partly overlapping. The structuring mechanisms that define the coupling between languages can be used to translate a specification into a different representation which is more suited for the task to be undertaken. An example of overlapping languages in TEMPORA are the PID and the ERL. The coupling between these languages was defined in Chapter 5 and it can be used to automatically generate ERL skeletons from a the PID specification. In this way, we exploit the visual features of PID to aid the more complicated task of modelling of rules. We will use an example to show how conceptualisation of rules can be supported by using structuring and filtering mechanisms.

We want to specify the rules that describe the entering of the delivery note head for a delivery note, i.e., we focus on the process P6.1.2 Enter delivery note head in
Figure 8.5. This diagram also contains information about the other processes which are irrelevant for the formulation of rules for P6.1.2 Enter delivery note head. A better choice is therefore to use the viewspec in Figure 8.12 which contains only the relevant details.

To specify rules we need to know how the flows entering and departing from process P6.1.2 are related. This is achieved by updating the viewspec in Figure 8.12 to include ports. The result is shown in Figure 8.21. This diagram is used as a basis for rule specification. Using the formal definition of the coupling between the process model and the rule model, the following rule skeleton can be derived from the diagram:

T1:

\[
\text{when enter\_dn\_head} \\
\text{if dn\_info} \\
\text{then dn\_info and enter\_dn\_head\_ok and} \\
\quad (\text{enter\_dn\_line\_ok or status or (status and enter\_dn\_line\_ok)})
\]

T2:

\[
\text{when modify\_dn\_head} \\
\text{if dn\_info} \\
\text{then dn\_info and enter\_dn\_head\_ok and} \\
\quad (\text{enter\_dn\_line\_ok or status or (status and enter\_dn\_line\_ok)})
\]

The relation between the combinations of input and corresponding output of the process cannot be determined from the diagram, only the relationship between input flows and relationship between output flows. As described in Chapter 5, the two composite rules T1 and T2 can be elaborated into more precise rules when we take into account the values of the different flows and the process logic. In this case it results in four different rules, R206121, R206122, R206123 and R206124. These are depicted in Figure 8.17 and they are specified in detail in Appendix E. When the number of values becomes high, decision tables have shown to be useful in order to formulate rules for the different cases. The use of decision tables is beyond the scope of this text and we refer the interested reader to [142] for a further description.

**Example 3: Documentation** In many cases, it is also useful to produce composite viewspecs which may consist of components expressed using different languages, e.g., the PID viewspec in Figure 8.12 and the ERT viewspec in Figure 8.14. Thus, a composite viewspec consists of two or more simple viewspecs, where each is generated as specified above. All kinds of viewspecs can be referred to, retrieved, manipulated and stored. The viewspecs are independent specifications and thus persistent viewing is supported (see below). This gives the user flexibility to produce viewspecs according to the her wishes, e.g., to interleave rules and diagrams in a report. In
Section 8.4, we will show that persistent viewing is necessary for using viewspecs as a basis for entering new information into specifications. Composite viewspecs may also consist of subviewspecs that are located at different specification levels, e.g., rules at different levels.

Types of medium used for presentation Viewspecs can be produced interactively or by invoking a filtering function with the necessary information passed along and the result is returned to the user. As mentioned above, the suggested filtering mechanisms can be used for viewing both in a non-persistent and persistent manner. In the former case, the filtering mechanisms are integrated in an editor and thus correspond to advanced viewing facilities. The filters to generate viewspecs can be used interactively to investigate specific aspects of the model (i.e., displayed in windows on the screen) or the viewspecs can be included in a report, stored in separate files, etc. This is also the case in the case of persistent viewing but in addition the viewspecs can be dealt with as independent specifications and thus they can be stored and used for viewing at a later time or used as input for further modelling like any other specification generated by an editor. As we will see below, persistent viewing is a prerequisite for allowing updates of viewspecs. How non-persistent and persistent viewing can be realised in a tool environment is elaborated upon in the next chapter.

Co-Existence of Multiple Viewspecs

From the ERT, PID and ERL specifications we have generated a set of viewspecs which we would like to be able to work with simultaneously, i.e., we want to allow multiple viewspecs to co-exist. To use multiple viewspecs effectively in the modelling
process requires that:

- *Management* of multiple viewspecs is supported.

- It is possible to go *back and forth between specifications at different abstraction levels*, i.e., flexible ways to go back and forth between full specifications and associated viewspecs.

To achieve this a versioning and configuration management system is needed to keep track of specifications. Appendix B provides a brief description of the main features of a system for versioning and configuration management of specifications. It is developed by Andersen [3]. As far as versioning of viewspecs is concerned, the system described in [3] was developed when our work was in a rather immature state. It has therefore been necessary to adapt the solution in [3] to fit better with the approach based on complexity reduction suggested in this thesis. Previous work has been modified and extended in the following directions:

- Clarification of terminology as far as viewspecs are concerned.

- The set of relations which were introduced in [3] to deal with different types of viewspecs are revised.

- The concept of local workspaces is introduced.

- Description of how to use viewspecs systematically in the modelling process is provided. The focus in this thesis is more on how versioning can be exploited in conceptual modelling to construct models rather than on the management of specifications as such.

Details about the versioning of viewspecs are only included to the extent it contributes to illustrate the complexity reduction approach. The interested reader is referred to [3] for a detailed description of versioning and configuration management.

**Management of multiple viewspecs** Figure 8.8 showed different relationships between specifications. It shows that we must keep track of how the viewspecs are related to their originating specification as well as to other viewspecs. To allow multiple viewspecs and multiple viewspecs to co-exist, we distinguish between three different relations in the versioning graph: filter, variant and context relations.

A filter relation depicts the relationship between a viewspec and its originating specification. Two specifications are filter related if one specification is generated from the other specification by a filter. Thus, a filter relation depicts relationship between a full specification and a viewspec or between two viewspecs. This corresponds to A and B in Figure 8.8, respectively. Moreover, a specification can be filter related
to several viewspecs, that is, several viewspecs are generated from the same specification. Figure 8.22 shows a versioning graph that depicts a situation where a transaction has checked out a specification $S1.1$ and a set of viewspecs have been generated from it. We use the naming schema developed in [3] with some modification as we will describe below. A specification is uniquely identified by name $S1.1\{\tau\}(\alpha), \lambda$, where $\{\tau\}$ indicates it belongs to transaction $\tau$, $\langle\alpha\rangle$ indicates that it is a viewspec of type $\alpha$ and $\lambda$ is the revision number local to the transaction.

Figure 8.22: Viewspecs are filter related to their originating specifications.

A filter relation corresponds partly to the parallel relation in [3], i.e., to the filter relation between a full specification and a viewspec. However, a corresponding relation to the filter relation between two viewspecs is not addressed in [3]. We have chosen to change the name of this relation to achieve uniformity among the relations and also as we will see below, to achieve uniformity between versioning of full specifications and viewspecs. Thus, we must also extend the naming schema accordingly as is shown in Figure 8.22. If we introduce versioning to our example in Section 8.1, it can be represented in a component structure graph as shown in Figure 8.23. We will limit ourselves to show how the versioning system can be used for the ERT and PID specifications presented in this chapter. In the previous sections we have showed different ways of generating viewspecs for the PID and the ERT specifications. Figures 8.24a and b show how this can be depicted in version graphs for the ERT and the PID, respectively. From the ERT specification in Figure 8.3, we have generated three different viewspecs (Figures 8.14, 8.15 and 8.16). In Figure 8.24a these are named $E1.0\{1\}(1).1$, $E1.0\{1\}(2).1$ and $E1.0\{1\}(3).1$, respectively. Let us say we have used the viewspec in Figure 8.14 as input to two different filters, e.g., an exclusive timestamping filter and a constraint filter. This is depicted in the version graph as shown in Figure 8.24a, where they are called $E1.0\{1\}(1).1$ and $E1.0\{1\}(1.2).1$, respectively. Similarly, Figure 8.24b shows how the PID specification of P6 Register delivery note (Figure 8.5) and the associated viewspec in Figure 8.12.
Figure 8.23: The component structure graph for the ERT and PID specifications.

Figure 8.24: a) The ERT specification and associated viewspecs b) the PID specification and associated viewspecs.
8.3. Presentation

We make a distinction between updates of a viewspec that concern the contents, i.e., change of semantics, and updates that concern the representation of a viewspec without changing the semantics, e.g., restructuring of diagrams. In a similar manner as for full specifications, it is useful to talk about revisions and variants of viewspecs. An updated viewspec is a revision of its originating viewspec, whereas a viewspec is a variant of another viewspec if it is expressed using a different language or its layout is modified but in both cases the semantics of the specifications remain unchanged. We postpone the discussion of updates of viewspecs until the next section. A variant of a viewspec is denoted by a variant relation. This seems logical since restructuring is a modification of a diagram without changing its semantics, which is exactly what the concept of variant denotes. The layout is the only difference between a restructured viewspec\(^4\) and the original one. Accordingly, we are allowed to keep different variants of the layout of a viewspec. It is up to the analyst to decide how many layout versions of a viewspec she actually wants to keep. In Figure 8.24a we did ignore that the viewspec in Figure 8.16 has been restructured due to relaxed space constraints after the inclusive generalisation filter has been performed. If we want to include the intermediate viewspec that is the viewspec which is the result of the filter before modification of layout, we indicate this in the versioning graph by relating the viewspec and the restructured viewspec by a variant relation (Figure 8.25). The \(L\) indicates that the version is the result of changing the layout of the viewspec without changing its semantics.

![Figure 8.25: Viewspecs are variant related to their originating specifications.](image)

When working with a hybrid language like TEMPORA, it is useful to be able to have an easy access to related viewspecs expressed using a different language. This is provided by introducing the context relation. A viewspec \(V1\) is context related to another viewspec \(V2\) if \(V2\) describes related aspects of \(V1\), i.e., \(V2\) elaborates the context modelled in \(V1\). An example is a PID viewspec (e.g., Figure 8.21) and an ERT viewspec (e.g., Figure 8.15). The latter shows the objects referred to by the processes in the PID viewspec. The viewspecs are closely related and the developer may go back and forth between the viewspecs, e.g., when formulating rules. To support this way of working we therefore relate the viewspecs in Figures 8.21 and 8.15

\(^4\)The same is true for a specification which is translated into a specification represented in a different language.
by a context relation in the versioning graph (Figure 8.26). $P1.0\{1\}\{1\}.1$ corresponds to the viewspec depicted in the version graph in Figure 8.24a.

![Diagram](image)

**Figure 8.26: Viewspecs are context related to their originating specifications.**

**Back and forth between specifications at different abstraction levels**  It should not only be possible to allow multiple viewspecs to co-exist but it should also be possible to go back and forth between specifications at different abstraction levels in a flexible manner. This implies that it should be possible to go back and forth between a full specification and its associated viewspecs as well as different versions of such specifications. To keep track of all the specifications we have already introduced some relations for depicting relationships between specifications. In addition, we introduce the notion of local workspaces. The purpose of local workspaces is to hide irrelevant specifications, provide easy access to relevant specifications and provide a more comprehensible naming schema for relevant specifications. Practical usage of the viewspecs have shown that the number of viewspecs associated with a large specification may become large and its difficult to keep track of all of them.

Although the definition of transaction can be considered as a local workspace, the introduction of a flexible filtering mechanism calls for a finer granularity of workspaces than is provided by development transactions in [3]. Thus, we suggest that a transaction can be divided into a number of local workspaces. These are defined by the developer according to her needs. Each local viewspec may consist of a set of full specifications and a set of associated viewspecs, i.e., a subset of a development transaction. Figure 8.27 shows an example of how local workspaces can be denoted in a versioning graph. The selection of viewspecs to participate in a particular workspace is a user-decision.

The advantages of providing the user with the ability to define her own workspaces are the following:

- Easy to keep an overview of relevant specifications for the task to be undertaken at any time during the development process. The relevant viewspecs can be shown by a graphical facility which provides overview pictures of global and
local workspaces. Such a facility would utilise the information already provided in the versioning system. The use of local workspaces may resolve the conflict of providing a sensible context for carrying out the actual work. Thus, the concept of complexity reduction also applies to versioning.

- Easy to go back and forth between different abstraction levels. Advanced user interface packages facilitate such a working mode by allowing several windows to be displayed at the same time, cut and paste between windows, *etc.*

- The introduction of local workspaces allows us to have local name spaces for viewspecs. A more flexible naming schema is necessary because it is simply too inconvenient to deal with names of the type suggested in [3]. One should not be disturbed by transaction name and full specification name. The name of a version of a viewspec must be unique within the local workspace but different workspaces may contain common names. A component of a local workspace can be uniquely identified by prefixing the component name by the unique name of the checked-out originating specification and the local workspace name. Thus, the system should work with surrogates internally which uniquely identify specifications. However, the developer only deals with simpler name within each local workspace.

Some of these ideas can also be extended to encompass versioning of full specifications. However, this is beyond the scope of this work and is not dealt with.

### 8.4 Entering New Specification Details

This section details how viewspecs can be used as a basis for entering new information into specifications.
Why Allow Updates of Viewspecs?

By allowing updates on viewspecs we may use them as a basis for entering new information into specifications as well as for presentation purposes. The rationale behind allowing viewspecs to be updated is:

- *Large and complex specifications are difficult to update.* Rather than entering new information to a large and complex specification, the developer can concentrate on relevant details by updating appropriate viewspecs generated from a full specification. The developer is not confused by irrelevant details and thus, modifying a smaller and less complex specification requires less effort.

- *Explore alternatives.* In some situations, the actors envisage several alternative solutions which may be interesting and important to pursue. The use of viewspecs allows different solutions to be modelled and it also provides support in managing the resulting specifications.

- *Conflicting views of reality are useful to model as intermediate results.* Different parts of a model may be based on slightly different and inconsistent views of the reality, e.g., different actors may hold slightly different views of the reality and the system to be built. To resolve potential conflicts among these views, it may be useful to model these as intermediate results. In some way, the final information system must accommodate a compromise of all these views.

- *Generate appropriate viewspecs for presentation.* We have defined a set of filters. The user may however want to generate a viewspec for which no filter is defined, e.g., the resulting viewspec may still contain irrelevant information which must be suppressed manually.

The importance of allowing viewpoints to be updated has also been pointed out by other researchers, e.g., Easterbrook [29].

Management of Updated Viewspecs

We have mentioned before that revisions of full specifications has a counterpart in updated viewspecs. Figure 8.28 shows a version graph for a full specification with associated viewspecs. Viewspecs are updated and this is depicted by a revision relation. It is the contents of the viewspec that are changed, e.g., by adding or deleting information. Each update of a viewspec results in a new revision of the viewspec. The update of the viewspec in Figure 8.12 to include ports (Figure 8.21) is depicted in the version graph in Figure 8.29.
The Problem of Inconsistency

Allowing viewspecs to be updated implies that they can be modified *directly* by the developer. It is up to the developer to manipulate the viewspec and therefore, no control is to be provided by the system beyond consistency checking which is also performed for full specifications. However, by allowing updates on viewspecs another type of inconsistency problem also arises. Updating a viewspec may imply that the updated viewspec and the full specification from which the viewspec originates, become *inconsistent*. The inconsistency may occur for several reasons. Updated viewspecs may have (relative to the originating specification from which they were generated):

- New components.
- Updated components (slightly modified components).
- Deleted components.

Conflicts may occur such as conflicting names, conflicting types and conflicting use of domain concepts in different specifications. Also conflicting views of the reality can be explicitly modelled. In the next section, we will address how such conflicts can be detected and solved.
Alternative Ways of Dealing with Inconsistency

In spite of the problems of inconsistencies, we have chosen to allow updates on viewspecs because we want the viewspecs to be used 'actively' during the development process. It is crucial that the developer can enter new specification details into viewspecs. The developers can thus concentrate on relevant issues without being disturbed by irrelevant details which is essential when dealing with large and complex specifications. The price we have to pay for allowing updates of viewspecs is to provide support to assist the developer in dealing with inconsistent specifications. We envisage at least three solutions:

1. Allow updates of viewspecs but insist on that inconsistencies should be resolved before any other action can be taken. This could be done by creating a skeleton, i.e., a placeholder, for a new revision of the full specification as soon as a viewspec is updated (Figure 8.30). The conflicts between the viewspecs and the originating specification must then be resolved before the development can proceed.

2. Allow updates of viewspecs and allow for inconsistencies to exist but provide extensive support for the eventual specification merging process (Figure 8.31).

3. Allow updates of viewspecs and allow for inconsistencies to exist but leave the specification merging process to the developer.

![Diagram](image)

**Figure 8.30:** Resolving the conflict immediately after a viewspec is updated.

The first alternative corresponds to a rigid development approach, where it would not be allowed to have several updated viewspecs to co-exist. This contradicts our requirement to a flexible approach. If we go for the third approach we only need to provide versioning of updated viewspecs and leave the process of integrating the new versions into the full specification to the developer. This does not seem to be the way to go due to the fact that providing this flexibility may do more harm than
good in the development process. In particular, this alternative may become quite
time-consuming when many viewspecs are updated and used as a basis for updating
a full specification. We have to come up with facilities to support the process of going
from one full specification to another one by taking into account updated versions
of viewspecs. Thus, the second solution is what we will go for in the remainder
of this chapter. The next section outlines support for the process of building full
specifications based on a set of updated viewspecs.

8.5 Inclusion of Changes

This section details how changes that are made to one or more viewspecs can be used
as a basis for building a revised version of a full specification. To carry out the process
of including the changes contained in the viewspecs into the full specification, is not
straightforward. During the modelling process several viewspecs may be updated
and several versions of each viewspec may co-exist. Thus, an important step in a
modelling process based on viewspecs is to provide support for the process of solving
eventual conflicts between updated viewspecs and the originating specification. We
distinguish between two major ways of including changes:

- Change propagation.
- Specification integration.
Chapter 8. The Complexity Reduction Approach

Controlled Change Propagation

The simplest way of including changes on updated viewspecs into the respective full specifications is to propagate the changes one by one. Changes to a specification are not by default propagated automatically. For large specifications the number of interconnections is usually very high and the consequences of the changes may not be predictable, e.g., an update of an entity class in the ERT may cause several changes to ERL rules that contain ERT expressions that refer to the entity class. To propagate changes in a controlled manner we advocate an approach based on a mixture of automated support and manual intervention. In general, changes to a viewspec $V$ are only considered for propagation if the viewspec is selected by the developer. For example, if a viewspec is updated and only used for presentation purposes, we do not consider to propagate the changes.

An updated viewspec may contain modified objects that should also be propagated to specifications other than the originating specification. By using the links that define the interrelationships between the languages (cf., Chapter 6), a list of affected objects in associated specifications can be provided. Thus, search facilities that utilise intra-language, inter-language and inter-level links are valuable support when changing/modifying a specification\textsuperscript{5}.

Warnings could also be provided to inform the developer that other associated concepts are affected and must be updated in order to maintain consistency across specification boundaries. Then the developer may choose whether she wants to propagate a change or not. The changes are carried out one by one and some may also be propagated automatically.

In some cases, automatic propagation of changes may be appropriate but it should be user-driven and only performed if one can assure that it has no undesired side-effects. That is some on-off mechanism should be provided. This can be illustrated with a simple example. Let us say that we want to change the name of an entity class from dept to department.name. It is obvious that before the change is propagated the developer should be provided a warning if department.name already exists in the specifications. If department.name does not exist the change can be propagated to all specifications, going across specification boundaries using appropriate links.

Specification Integration

The process of putting a new revision of a full specification together based on viewspecs will be referred to as the specification integration process. The result of the specification integration process is a consistent version of the previous full specification including the changes in the selected viewspecs.

\textsuperscript{5}This is implemented in the TEMPORA capture tool which provides automatic generation of documentation/references to documentation with regard to inter-model issues.
A thorough analysis of specification integration is beyond the scope of this work. We limit ourselves to describe major steps of the integration process and to outline some means to support such a process. The support itself will be a number of very specialised tasks, where each task in itself may be a doctoral thesis!

**The input to the specification integration process** Figure 8.32 shows how specifications that form the basis for a specification integration process is depicted in a versioning graph. The input to the integration process is related with *merge* relations between the relevant specifications. Possible specifications are:

- A previous version of the full specification.
- One or more update viewspecs.
- One or more viewspecs which are approximations of the corresponding originating specification.

Additional information may be provided about each updated viewspec, e.g., type and what changes have been made since it was created. This information is recorded during the filtering and updating processes. The versioning facility provides the information together with the names of the specifications. The specifications are then retrieved from the specification repository.

![Specification Integration Diagram](image)

**Figure 8.32: The merge relations.**

Only viewspecs which contain approximations of the originating specification and updated viewspecs should be considered for integration. Other viewspecs are not relevant because they are projections of the originating specification and thus, the viewspec’s information is already included in the full specification. However, not all updated viewspecs are considered for integration. It is a *user decision* to select what viewspecs that should be considered for integration. For example, if a viewspec is...
updated and only used for presentation purposes, we do not consider to propagate the changes. We may also have situations where updated viewspecs are used to explore alternative solutions/conflicting viewpoints. Only the set of viewspecs that represent the 'consensus' solution are considered for the integration process. In this case, we envisage that additional facilities are provided to support the process of arriving at a consensus solution, e.g., [70].

The viewspecs used as a basis for the specification integration may contain (relatively to the full specification from which they originate): new components, updated components, deleted components, and/or redundant components. If an update of a viewspec only results in redundant components the updated viewspec does not need to be considered any more in the integration process. If components are deleted from the viewspec, corresponding components in the full specification must be considered for deletion. If new components are added and they do not exist in the specification, they must be considered for insertion in the new version of the specification. Updated components implies that the corresponding components in the full specification must be considered for modification.

**Outline of the specification integration process** Specification integration correspond very much to view integration which has been a topic of much research in the database community. It can be considered as consisting of four subtasks\(^6\) [9]:

1. **Preintegration.** When several viewspecs are used as input to viewspec integration, one must decide on how many viewspecs should be considered at a time. A number of strategies have been developed in the database community to support viewspec integration, such as [31]: binary ladder integration (i.e., the two most similar viewspecs are integrated first, etc.), N-ary integration (i.e., all the viewspecs are integrated at once), balanced binary strategy (i.e., pairs are integrated in a balanced tree) and mixed strategies. The choice of strategy is crucial to avoid the problem of information overload, where the persons performing the integration have too many overlapping specifications to integrate at once.

2. **Schema comparison** includes identifying correspondences and detecting conflicts among the viewspecs. Types of conflicts that may be detected are [31]: naming conflicts (e.g., synonym and homonym), type conflicts, domain conflicts and constraint conflicts.

3. **Schema conforming** aims at solving the previously detected conflicts. Representation of the same concept in two different specifications can be classified as follows [40]: identical, equivalent, compatible and incompatible. To deal with such conflicts traditional approaches are mostly based on either transformational equivalence or they entrust the skill of the designer by providing him only examples valid for the particular model [40].

\(^6\)These steps have become the *de facto* standard for view integration in the field of databases
4. *Merging and restructuring*. The different specifications are merged into a global specification and then restructured. The latter involves testing that the resulting specification agains a set of criteria such as completeness, correctness, minimality and understandability [40].

A more detailed description of these steps and how they are supported by different approaches are found in [40].

Most likely viewspecs contain less information than the corresponding full specification. It may therefore be an advantage to do it in two steps because of the size of the specifications involved:

1. *Viewspec integration*. If there are more than one related viewspec that forms the basis for the specification integration process, the viewspecs themselves should first be considered for integration. Overlapping viewspecs are integrated whereas independent viewspecs remain separate.

2. *Integration the viewspecs and originating specifications*. Integrate the resulting viewspecs from step 1 with the full specifications from which the viewspecs originate.

This is only relevant if one knows that the information in the viewspecs are not already in the full specification. In addition to consistency checking, the versioning system together with additional information from the filtering and updating processes can provide useful information about the viewspecs. The versioning system sets each viewspec in the context of the originating specifications and associated viewspecs. Thus, by providing additional information a step-wise process of viewspec integration can be supported.

**Tool support for specification integration**

The ideal situation would have been to allow updates on viewspecs and provide enough support to perform an automatical merge of relevant specifications. However, we cannot expect this process to be fully automatic but rather a mix of manual intervention and comprehensive computer support:

- *Manual interventions*. The most general way of working will be to inspect the specifications and decide what to integrate in the revised version. Based on this inspection relevant information may partly be manually extracted and integrated into a new full specification.

---

7Each step involves the integration process as outlined above.
Computer support. The computer support encompasses simple support of the manual interventions (e.g., 'cut and paste' between specifications) to more advanced support (e.g., integration techniques that utilise the semantics of a specification).

Specification integration is a complicated matter and the integration process may be carried out in different ways depending on factors such as the language which is used to express the specification, the size of the specification and the number of people who are involved in the integration process. In addition, to the classical approach taken by most approaches from the field of data modelling, more advanced approaches are now being developed which utilise recent advances in other fields such as object-orientation and knowledge based systems. It is beyond the scope of this text to give a survey of such approaches and we refer the interested reader to [40] for a description of trends in approaches to specification integration.

To conclude our section on specification integration we will briefly mention some means which may be relevant to investigate for our approach. Integration of specifications is dependent on what kinds of viewspecs that are going to be merged. Certain properties of the merging process are similar irrelevant what languages that are used in the viewspecs (language independence). More advanced support can be provided by taking language dependent features into account and also exploit recent advances in networking and workstation technology. We envisage various ways of supporting the merging of specifications such as:

Computerised support of manual integration. Manual merging may be performed in various ways by exploiting modern user interface technology. Thus, a variety of working styles can be supported such as virtual paper, clipboard technique between specifications, ordinary 'cut and paste' between specifications and 'active' structures.

We may also provide facilities to keep track of changes done to a viewspec after checkout from a full specification, i.e., by recording modelling history during the updating and filtering processes. The changes could be recorded textually (e.g., the user get a list of operations and objects upon which the operations are performed in a separate window) or shown explicitly relative to the components in the corresponding full specification, e.g., shown with a particular notation in a diagram. In the latter case, cut and past facilities across windows, i.e., between specifications, greatly improve the merging process. In addition, the versioning system may provide information which is acquired during the filtering processes such as what filter a viewspec is generated by and when this was done.

More advanced integration support. Automatical merge of specifications is mostly limited to a syntactical merge of a set of specifications. The result of such a merge is useful only in some cases, e.g., if all components of the different specifications have got unique identifiers. However, the use of formal specifications opens up for more extensive specification integration techniques
where for example structural conflicts may be resolved. This is useful in most modelling situations where different actors in the development process may have different perceptions of the domain (and the target information system) and consequently, this leads to different representations. To solve such conflicts a number of conflict resolution techniques are proposed, e.g., [97, 62]. An overview of approaches for view integration is found [40].

- **CSCW (computer supported cooperative work) support.** We may also envisage a cooperative approach for integrating viewspecs with the originating specification where several people are involved in the process. In this way, consensus about the new revision of the full specification is reached by active participation of all the actors involved in the modelling process. The integration effort may take place as a face-to-face meeting or conducting by applying more advanced workstation and networking technology. New workstation and networking technology allows people to work in their familiar environment and work simultaneously on the specifications.

The types of support outlined above represent different ambition levels and these may also be improved by combining the various features in novel ways.

### 8.6 Complexity reduction in the TEMPORA Development Process

The complexity reduction approach suggested in this chapter is not a systems development approach as such but it can be used together with a development approach to enhance the latter’s abilities to deal with large and complex systems. In [113, 137] we have shown how complexity reduction can be applied in the TEMPORA approach by suggesting a TEMPORA development discipline including the complexity reducing features suggested in this work. It describes how the complexity reduction approach can be used in the various phases of the TEMPORA development process and it also provides numerous methodological guidelines for how to use the different complexity reducing features, e.g., guidelines for how to use the different filters. Rather than repeating the development discipline here we refer the interested reader to [113, 137] for a detailed description.

### 8.7 Chapter Summary

This chapter has described the main activities of the complexity reduction approach: filtering, presentation, entering new specification details and inclusion of changes. We have detailed each of the activities and suggested how they can be supported. The latter activity has however only been briefly addressed since we consider spec-
The complexity reduction approach can be used for building specifications from scratch, from existing specifications or any combinations of the two. The only assumption is that at some point in the modelling process, the number of details reaches a stage where the comprehensibility of the overall system drops substantially. In other words, the dilemma caused by the contradictory roles of a specification becomes clear. The inclusion of more details is necessary but at the same time this will blur the picture even more making the specification less appropriate as a basis for communication and understanding.

Our approach to balancing the contradictory roles is to facilitate integrated specifications but at the same time provide facilities that allow the developers to split a complicated task into a set of more comprehensible subtasks. Integrated specifications are facilitated through the structuring mechanisms described in Chapters 5 and 6. By providing the notion of viewspecs and filters to generate the viewspecs as defined in the previous chapter, details can be separated in a systematical manner, enabling developers to concentrate on relevant parts of the system at a time.

The possibility of using pre-defined filters, user-defined filters, and combinations of these provides the developer using the complexity reduction approach with a great deal of freedom in focusing on different aspects of a system. Rather than operating on a full specification, relevant viewspecs can be applied at different stages of the development process, e.g., in rule capture. The viewspecs are used ‘actively’ during the development in the sense that they do not only serve as passive components or projections of a set of specifications (e.g., for presentation purposes) but also serve as a basis for entering new information into specifications. This is achieved by allowing viewspecs to be updated and and then eventually integrated into a new version of the full specification. By allowing viewspecs to be updated, both top-down modelling and bottom-up modelling are supported. Furthermore, multiple viewspecs are allowed to co-exist. To glue the contradictory actions of integration and fragmentation, additional support is provided to allow the developer to go back and forth between the integrated specifications and their associated viewspecs. In particular, we have adapted the solution to versioning and configuration management in [3] to fit better with the approach based on complexity reduction.

The complexity reduction approach is not a systems development approach as such but it can be used together with a development approach to enhance the latter’s abilities to deal with large and complex systems.
Chapter 9

Applying Complexity Reduction in PPP

The work reported so far in this thesis has been conducted and targeted to solve problems related to development of large systems using the TEMPORA approach. In Chapter 5, we showed how the process based and rule based approaches can be combined. The Chapters 6 to 8 proposed means to deal with complexity in the systems development process. To show how these solutions can be exploited for the PPP approach we will specify how they can be included in the PPP CASE tool. The revised version of PPP is under development and thus, the features suggested in this work are not yet implemented.

Section 9.1 presents the PPP approach. Sections 9.2 and 9.3 detail how the PPP approach can be extended to deal with rules as well as complexity in the modelling process.

9.1 The PPP Approach

The PPP approach comprises a set of languages, methods and tools to support information systems development from initial problem description and requirements specification to implementation and maintenance. This section presents the PPP languages and the implementation of the PPP CASE tool.

The PPP Languages

The PPP languages are: the Phenomenon Language (PhM) [122, 123, 93, 157], the Process Language (PrM) [10, 93, 47] and the Process Life Description (PLD) [152]. The basic constructs of the PPP languages are adopted from well-known and widely
used modelling languages. The PhM is an extended ER language and describes the static aspects of the UoD. The PrM and the PLD describe the dynamic aspects and are based on Structured Analysis [26] and graphical program specification [144], respectively.

**The PhM**  The PhM is an extension of the traditional Entity-Relationship language. The basic modelling constructs of PhM are: entity classes, relationship classes, attribute relations and data types. The PhM distinguishes between data entities and non-data entities. The non-data entities describe real world objects whereas the data entities describe the informational counterparts, i.e., information about the real world objects. An example of a PhM model is shown in Figure 9.1 and as we see, the graphical notation is slightly different from traditional ER notation.

![Figure 9.1: An example of a PhM model.](image)

Entity classes, relationship classes and cardinality constraints have the same meaning as in the ER. To increase the expressive power, the PhM contains the following extensions compared to the traditional ER: attribute relations, relationship constraints, generalisation/specialisation hierarchies, conceptual views and a PhM algebra. For more comprehensive descriptions, the interested reader is referred to [120, 122, 123, 93, 157].

An **attribute relation** links a data type to an entity. Attribute relations are classified into four types [124]:

- **Identifier** which uniquely determines instances of an entity class (denoted `id`).
- **Attribute** which is used to denote single-value properties to instances of classes (denoted `att`).
- **Repeating group** which is used to denote multiple-value properties to instances of classes (denoted `rep`).
9.1. The PPP Approach

- Quality which is used to denote properties that are characteristic of a class rather than of the elements of the class (denoted qual).

A relationship constraints specifies if an entity class participates fully or partially in a relationship. An entity class relates fully (denoted f) in a relationship if all the instances of the entity class participate in the relationship; otherwise it relates partially (denoted p) in the relationship.

Generalisation is supported through the subclass concept (denoted S). An entity class may have several subclasses. All the members of a subclass are also members of the subclass' superclass and they inherit all the properties of the superclass.

Conceptual views\(^1\) are introduced to allow definition of different views of a domain, i.e., each conceptual view consists of a collection of entity classes, relationship classes, etc. They can be used in a similar manner as how SQL views [25] are used to define various views of an SQL application. A PhM model can thus be extended or modified by adding new conceptual views. By providing an algebra, extensons of the model can be done by defining new components, e.g., entity classes, in terms of already defined components.

The ONE-R algebra is developed by M. Yang [157]. It is based on relational algebra but it is extended to manipulate components in a PhM model. The ONE-R algebra provides the following operators: union, difference, product, selection, projection, reduction, nest, unnest, pack, unpack, join, natural join, relationship join and packed relationship join.

The PrM The PrM is the origin of the PID already described in Chapter 3. The basic modelling concepts are processes, stores, external agents, flows, ports, buffers, items, resources and timers. An example of a PrM model is shown in Figures 9.2 and 9.3. We will not repeat the description of the PID here but we rather outline some of major differences between the languages:

- The concept of data store in the PhM corresponds to the ERT view construct in the PID.
- The concept of terminating flows, resources and buffers are only supported in the PhM. Terminating flows indicate the termination of a process' execution. A resource contains items that are necessary for a process to run. A resource may be connected to several processes and items are retrieved and returned by the processes. Triggering and non-triggering items may be buffered upon arrival at a process class if there is no process instance to consume them. Such items are stored in a buffer. Items, buffers and resources are not depicted in a diagram.

\(^1\)A conceptual view was initially called a scenario [121].
• The graphical representation of external agents, triggering flow and data store is different (compare Figure 9.2 and 3.3).

For more detailed descriptions of PrM, see [10, 93, 47].

![Figure 9.2: Top-level PrM model for oil processing design.](image)

![Figure 9.3: Decomposed PrM model for oil processing design including ports.](image)

The PLD The PLD is a procedural language and it is used to describe the process logic of non-decomposed processes. The constructs of the PLD:

- **Start**: indicate beginning of a PLD specification
- **Receive**: receive dataflow
- **Send**: send dataflow
- **Assignment**: symbolise a block of program statements or a subprogram call
9.1. The PPP Approach

- Selection: IF-test
- Loops: FOR and WHILE-loops

An example of a PLD specification of process P1.3 Choose Separator Condition in Figure 9.3 is shown in Figure 9.4. When the ports are designed in a PPP diagram, the skeleton of the PLD is generated automatically, taking care of the reception and sending of data. Further design details can then be added. For a detailed description of the PLD, see [152].

![Diagram]

Figure 9.4: An example of a PLD specification.

The PPP CASE Tool

A revised version of a prototype CASE tool environment for PPP is under development. Compared with the previous version [47] new features such as explanation generation, performance evaluation, and complexity reduction will be provided. A graphical representation of the current PPP implementation is presented in Figure 9.5, the various components of which we will detail by first describing the purpose of each module and secondly how each module is implemented.

Purpose of Modules

- Modelling editors serve to allow the creation and checking of the conceptual and design level models of the application being developed. The editors
facilitate creation, retrieval, modification, deletion and storage of models represented in the PhM, the PrM and the PLD.

- **PPP repository** holds a copy of the conceptual and design models in a database, allowing access to the information through a versioning facility.

- **Analysis support** provides verification and validation support. Verification includes syntax checking and checking of semantics of PPP models. Validation includes translation of PPP models to executable prototypes.

- **Explanation facility** provides means to explain phenomena represented in conceptual models. The explanation generation facility for the PPP offers the user a set of questions to be asked to increase their understanding. It can also take into account user characteristics and the context in which the questions are asked.

- **User interface editor** serves to allow the creation and checking of static and dynamic aspects of user interfaces of the application being developed. It is based on the features offered by graphical user interface technology.

- **Document design** serves to allow the creation and storage of documents with specific formats, e.g., reports, forms and sheets.

- **Runtime generation support** provides support for the generation of the runtime system as well as for the execution of systems developed using the PPP approach.
• **Sybase** is the RDBMS used by any application built by the PPP tools when it is executed on the PPP runtime system.

**Implementation of the Modules** The prototype is being developed on SUN Sparc workstations which run UNIX Solaris 2.3. C++ is used as the implementation language for most of the functional features. Exceptions are some of the modules which are already implemented separately in BIM-Prolog and are included in the new version of the PPP tool by using the C++/Prolog interface. The user interface is built using the commercial user interface packages Interviews and Unidraw. All the services provided by the various modules described below are accessed through this common user interface. The following list gives a brief description of the implementation of each module. The communication between modules is made using the local PPP repository.

• **Modelling editors** including the document design editor are built using Interviews, Unidraw, an C++. Interviews and Unidraw provide the graphical primitives necessary for building a graphical and form based CASE tool and is coupled with the PPP repository which acts as the persistent storage mechanism for the tools when in use.

• **PPP repository** holds a copy of the conceptual and design models in a database, allowing access to the information through a versioning facility. The repository also provides the means for integrating the various PPP tools through data sharing. The storage structure (also called the PPP repository structure) is partly\(^2\) derived from the PPP meta-model.

• **Analysis support** provides verification and validation support. Verification is to be provided as an integral part of the modelling editors. The user can choose the target language for the generation of executable prototypes.

• **Explanation facility** is implemented using Bim-Prolog.

• **User interface editor** is implemented using Interviews and Unidraw.

• **Sybase** is a commercial RDBMS which is used without modification as the RDBMS in the PPP (for both development as well as for the runtime system).

### 9.2 Complexity Reducing Features in PPP

This section describes how the complexity reducing facilities suggested in Chapters 5 to 8 can be implemented in the PPP CASE Tool.

\(^2\)The PPP repository structure must also convey some additional information about administrative matters, *etc.*
Areas of Exploitation

A number of problems faced in development of large and complex systems using the TEMPORA languages are also commonly found in development using the PPP languages [111, 73]. Moreover, the languages are also overlapping and complementary. This makes us believe that there is a potential for exploiting certain solutions already reported in this thesis for improving the PPP tool for large systems development. We will concentrate the exploitation effort to two areas:

1. The incorporation of rules in the PPP.
2. Extending the PPP to encompass complexity reducing features.

It may appear strange to suggest that rules should be included in the PPP approach to deal with complexity since rules as such increase complexity of systems development. The reason is however, that the lack of rule modelling in PPP is a severe problem in development of systems which are characterised by large collections of rules\(^3\), e.g., task processing systems. Thus, by extending the PPP with rule and the complexity reducing features already proved successful in TEMPORA, we intend to make PPP an ever stronger approach.

Modules to be Included

To extend the PPP architecture with facilities to support rule modelling and to provide complexity reducing capabilities, three modules must be added to the architecture:

- A *rule editor* which assists the developer in the rule modelling by providing conventional editing features, checking facilities, etc.

- A *viewing component* which allows the specification repository to be viewed from different perspectives. It also provides support for the generation of appropriate viewspecs for specification construction, validation and explanation generation. Other features such as support of manipulation of layout may also be added.

- A *specification integration component* which provides support of propagation of changes and integration of specifications.

The viewing and integration components will together be referred to as the Complexity Reducing Facility (CRF). As mentioned before, the details of the integration component are beyond the scope of this text and in the remainder of this chapter we will only detail tool support for the viewing component.

\(^3\)A detailed description is found in [111] which models the Oil Processing system outlined in Chapter 1.
Alternative Implementation Architectures

To add new components to the PPP architecture we may consider how tight or loose these components should be integrated. The rule editor is included in the PPP CASE tool in a similar manner as the other editors [158]. However, there are no similar components to the CRF and the integration of this component into the tool must be carefully considered. Obviously, there is at least three solutions for how it can be included in the PPP architecture:

1. The relevant functionality of the CRF is integrated with PPP components that use the particular services.

2. The CRF is included as a separate module but the services are available together with the other PPP services through the common user interface.

3. The CRF is included as a separate module which is invoked independently from the other components.

In the first alternative, the CRF is integrated with PPP components that use its services. Since CRF services may be requested by a number of components in an almost similar fashion this implies duplication of the code that implement the CRF services in the various components. Thus, this alternative would be tedious to implement and difficult to maintain. The architecture corresponds to the one shown in Figure 9.5 and does not show any of the complexity reducing features explicitly.

In the third alternative, the CRF is included as a separate module. This means that the services are not directly accessible from other PPP components. Since the services provided by the CRF will be used to support the major activities carried out in other PPP components, it may become awkward to invoke the CRF to carry out a particular task and then return to the invoking facility. The architecture where relevant components used at the conceptual level are included, is shown in Figure 9.6.

The second alternative provides a compromise by separating the integration of services accessed by various PPP components and the actual implementation of these services. We add the CRF as a separate module that provides various services to the other PPP components. The services are however, invoked from the particular PPP components that request the services. This means that the CRF is integrated with respect to services but the implementation of the service is not duplicated for each component. The architecture is similar to the one shown in Figure 9.6. In the remainder of this chapter we will elaborate on this architecture.
9.3 Incorporating Rules in PPP

This section suggests how the work reported in the previous chapters in the context of the TEMPORA languages can be exploited to enable incorporation of rules in the PPP.

How to Incorporate Rules

We have at least two possibilities for including rules in the PPP languages:

1. Substitute rules with PLD for describing process logic.
2. Allowing specification expressed using rules and PLD to co-exist.

The first option corresponds approximately to the TEMPORA languages. The only difference would be the use of PhM instead of ERT. Although TEMPORA has shown to be quite successful in dealing with rules, experience also shows that rules may be difficult to formulate, cf., experiences reported in Section 3.3. Therefore we would like to take the advantage of the user-friendliness of PLD for specifying procedural aspects and the advantage of the ERL for specifying declarative aspects. Thus, the second option is the most promising one from our perspective. A PrM process can be associated with: a set of decomposed processes, a PLD description, or a set
of rules. In doing this we achieve the following: The developer can choose what representation to use depending on how easy it is to express the information in ERL or PLD. The decision to express the internals of a process in a procedural manner can also be postponed to the design phase. Rules are expressed in a declarative manner at higher levels of abstraction, e.g., using ERL or subsets of ERL. Then as the specifications are gradually refined and a number of design decisions made, the specifications are expressed in a procedural manner, e.g., using PLD.

Furthermore, rules at various abstraction levels should be related to one another and also to other components expressed using the PhM and the PrM. This can be done in a similar manner to the approach suggested for the TEMPORA languages described in Chapters 3 and 6. The TEMPORA languages and the PPP languages have many common features. With regard to the languages for static modelling and process modelling, their way of viewing the world is conceptually similar. The process languages have approximately the same conceptual basis but differ in external representation. The ERT corresponds to the PhM although certain features of the languages differ. Thus, earlier work must be adapted as follows:

- The ERL must be adapted to use PhM instead of ERT.
- Determine if constraints should be expressed using ERL or ONE-R.

### Tool Support for Rules

To incorporate rules in the PPP CASE tool, a rule editor must be developed. It should be a syntax-oriented editor with checking facilities, etc. During the TEMPORA project considerable experience is gained in identifying desirable features for such an editor. Thus, the rule editor developed in the TEMPORA capture tool by SISU together with the suggestions for improvement provided in [139] should form the basis for the PPP rule editor. The development of the editor is not considered a task of this work, and a detailed specification is therefore omitted in this thesis. We will ourselves to specify necessary features to illustrate major features of filtering of rules in the next section. A rule editor is shown in Figure 9.7.

In addition, most of the other modules that already exist in the implementation architecture must be extended/modified to facilitate rule modelling, e.g., the explanation generation facility and user interface editor. We limit ourselves to describe how the PPP repository structure must be extended to deal with rules, and to describe a prototype implementation of the use of rules in the PPP.

**The PPP repository** The repository structure of PPP developed by M. Yang [158] is shown in Figure C.1 in Appendix C. The model is expressed using an extended PhM model. To accommodate rules as described above, the repository structure in Figure C.1 must be extended. Figure 9.8 shows an updated version of the relevant
Figure 9.7: The rule editor.

parts of the repository structure that are influenced by the introduction of rules. For simplicity reasons, the attribute relation type is omitted.

The prototype A prototype has already been developed which shows how rules generated by the TEMPORA capture tool can be used together with the PPP tool environment. A detailed specification of how this can be done is provided in a diploma thesis [66] which was supervised by the author and is also reported in [67]. However, the prototype was based on a previous version of the PPP CASE tool. The new version of PPP will support rule modelling at various specification levels by the inclusion of capture facilities for rules as outlined above.

The prototype integrates a former version of the PPP CASE tool (called PPM2001) and TEQUEL, the temporal rule manager. The latter is developed at the Department of Computing at Imperial College. The prototype is developed on SUN 3/60 workstations which run UNIX Sun OS 4.1. BIM-Prolog and PCE are used as the implementation languages.

In spite of practical limitations (e.g. the lack of a parser for ERL at that time) the prototype has demonstrated the feasibility of our approach. The approach seems very promising with regard to the coupling between processes and rules including dynamic as well as temporal aspects. We have implemented the following: Given a PPM specification, TEQUEL rules and C code are generated on which the temporal rule manager operates. Here it should be emphasised that both TEQUEL rules and C code are generated from the same initial specification and it is performed automatically. The coupling between PPM and the temporal rule manager is depicted in Figure 9.9.

TEQUEL rules are generated from the PPM structure whereas C code is produced
Figure 9.8: Updated repository structure that incorporates the use of rules in the PPP languages.

Figure 9.9: The coupling between PPM and TEQUEL.
from the PLD specifications. Referring to the pattern depicted in Figure 5.1, the if...then parts are implemented by external calls to C routines. The rule manager is extended (by adding a new predicate) in order to handle the processes implemented in C, and it is linked to a special interface to take care of the communication between C and BIM-Prolog. Appendix D provides an example of how we can use the prototype to generate executable code directly from the PPM and PLD diagrams.

9.4 The Viewing Facility

This section specifies how the structuring and filtering mechanisms for the TEMPORA languages can be adapted to be used with the PPP languages. It also describes the tool support for a viewing facility in PPP.

Adapting the structuring and filtering mechanisms

The structuring and filtering mechanisms in TEMPORA must be adapted to be used with the PPP languages. Fortunately, only minor changes are necessary. The PhM, PrM and PLD are integrated and a description is found in [47]. The PrM corresponds to the PID and thus the coupling as defined in Chapter 5 applies. Similarly, the filters for the PID and the rules which are described in Chapter 7 can be used without modifications. The connections between the PhM and the other languages must be slightly changed. The filters defined for ERT can also be applied however, certain modifications are necessary due to the differences in the languages. The process of building specifications based on viewspecs can however, be applied as described in Chapter 8.

PhM filters can be defined in a similar manner as the ERT filters. The filters will be slightly different due to some differences in the structure as shown below. However, the mapping itself is about the same. We suggest the following PhM filters: entity class filters, attribute relation filters, generalisation filters, cardinality constraint filters and component filters. These have all their correspondences in the ERT filters. The attribute relation filters correspond to value class filters, otherwise the names are the same. Rather than describing in detail each of the PhM filters we show with an example how the ERT value class filters can redefined to remove attributes from a PhM specification. The other filters can be defined in a similar manner.

The attribute relation filter can be formally defined as follows:

\[ F_{attribute\_relation} : V_{PhM} \rightarrow V_{entity\_class} \]
where

- \( V_{PhM} = (\text{Entity\_class}, \text{Relationship\_class}, \text{Is\_a\_relationship}, \text{Attribute\_relations}, \text{Cardinality\_constraint}) \)

- \( V_{\text{attribute\_relation}} = (\text{Entity\_class}, \text{Relationship\_class}, \text{Is\_a\_relationship}, \emptyset \text{Cardinality\_constraint}) \)

An example illustrating the use of the attribute filter is shown in Figure 9.10, where all the attributes in the PhM specification shown in Figure 9.1 are removed.

![Diagram](image)

**Figure 9.10:** An example of a PhM model where all the attributes are removed.

**Implementation issues**

The implementation of the viewing facility is similar to the implementation of most modules of the PPP CASE tools. Functional features of the viewing facility are to be implemented using C++, whereas Interviews and Unidraw are used for developing the user interfaces. The users access the services offered by the viewing facility through the common PPP user interface.

The formal descriptions of the filters in Chapter 7 provides both a definition of their data structures and the mapping from the data structure of the initial specification to the data structure of the resulting viewspec. The filters may be implemented using a procedural programming language or in a declarative language such as Prolog. In PPP they will be implemented using C++. To store the viewspecs in the PPP repository implies that the repository structure must encompass PPP construct extended with the specific concepts for abstract components in specifications. The viewing facility does not interact with the PPP repository directly but via the versioning facility. The influence of structuring and filtering mechanisms on the PPP repository and the management of viewspecs are separately dealt with below.
Impact on the PPP Repository

There are two factors that have an impact on the PPP repository, namely the introduction of specific concepts for abstract components in specifications and the extensions of the versioning facility to deal with viewspecs in a uniform manner.

Specific concepts for abstract components in specifications must be reflected in the repository structure. The viewspecs are simplifications of specifications expressed using the PPP language and thus, the PPP meta-model will contain all the constructs necessary to express all kinds of viewspecs except from the notions of abstract symbols, i.e., abstract ports and abstract flows. Thus, the domain from which the attribute relations can take their values from are extended with these types, that is, the attribute relation of type of Port must take another value, abstracted ports and the attribute relation of type of Flow must also take another value, abstract flows. These concepts must be added to the meta model of the PPP language, but because they only affect the domains of the attribute relations, the repository structure as depicted in Figure 9.8 remains the same. Beyond that the existing structure can be reused, however, certain parameters will become superfluous.

Management of viewspecs calls for inclusion of the concept of viewspecs in the version control system, as is already included in the extended version of the versioning of PPP specifications [3]. To support versioning, M. Yang has introduced the notion of versioned entity (Figure 9.11, adapted from [158]). However, to include the modifications as proposed in the previous chapter, the following changes of the repository structure with respect to the versioning facility must be made:

- Add more attribute relations for viewspecs which is important to record information about the filtering process, e.g., the filter type which is used to generate a viewspec.
- Add the notion of local workspaces.

Figure 9.12 shows an updated repository structure that incorporate the necessary changes.

Functionality of the Viewing Facility

In the sequel we will present some features of the viewing facility of the PPP CASE tool. Details about other PPP modules are only included when necessary to illuminate central issues of the viewing facility. We will present the viewing facility as the user will see it and carry out a modelling session in order to illustrate its layout
Figure 9.11: The definition of a versioned entity.

Figure 9.12: Updated repository structure that incorporate versioning of viewspecs.
and functionality. We will use some of the examples which were used in the previous chapter. Some central features of the viewing facility are explained throughout the modelling session and described with respect to what modules that request the services. However, services that are similar for several modules are not repeated for each module.

The viewing facility offers a set of services to the other PPP modules. The services may be accessed from any of the editors (the PhM editor, the PrM editor, the PLD editor and the ERL editor), the explanation facility, the validation facility, or the integration facility. An overview of the interaction between the viewing facility and other PPP modules is shown in Figure 9.13. The different lines between components correspond to procedure calls. Before the different services provided by the viewing facility are described with respect to each of the modules that uses the specific services, we briefly describe how the viewing facility interacts with the PPP repository.

![Diagram showing the interaction between the viewing facility and other PPP modules](image)

**Figure 9.13: Procedure calls from other components requesting services from the Viewing Facility.**

The interaction with the PPP repository The versioning facility is essential in managing viewspecs in the development process. The viewing facility does not interact with the PPP repository directly but via the versioning facility, i.e., via the data access functions (Figure 9.14, adopted from [3]). Viewspecs are created, stored, retrieved and deleted in a similar manner as full specifications. It is also desirable to reproduce a viewspec if the developer wants to go back to a previous viewspec. This may be achieved by storing the viewspec (i.e., in the case of manually produced viewspecs), by keeping track of how it was generated, or what kind of filter with which it was generated.
Figure 9.14: Refined PPP and CWM (cooperative work management) architecture.

The efficiency issues of concern are mainly how many viewspecs per full specifications that should be kept and how efficient these can be stored. At this stage it seems to be reasonable to say that viewspecs which can be derived from the full specification should be derived and not stored, whereas the others (such as restructured viewspecs) should be stored. Efficiency issues concerning the storage of the viewspec are mainly determined by the efficiency of the difference processor, e.g., UNIX diff and is therefore, taken care of by the versioning system. We refer the interested reader to [3] for a detailed description of viewspec management.

Services requested by the PhM editor Figure 9.15 shows the functionality of the viewing facility for the PhM (only the functions relevant for discussing complexity reduction are shown). As all PPP editors, the PhM editor is invoked from the common user interface. The user interacts with the tool through mouse handling, menu handling and the use of a keyboard. The window consists of three subwindows [73]: PPP command window, PPP concept window and PPP drawing window. The command window shows the major functions for manipulating a PhM model, i.e., the basic functions such as create a model, delete a model, etc., and the complexity reducing functions such as create a viewspec, delete a viewspec, etc. The concept window shows the concepts supported by the PhM model, whereas the drawing window is used to construct the model. A description of the general functionality of the PPP editors are found in [73, 158], and we limit ourselves to describe the features which are specific to the viewing facility.

As shown in Figure 9.15, the major functions are:

- **File** provides the following options: New (create a specification), Open (an existing specification), Copy (a specification), Save (a specification), Save as (gives the user the option to specify explicitly the file name), Delete (a specification), Print (a specification) and Defaults (environment variable is specified). A specification is either a full specification or a viewspec.

- **Filter** shows the pre-defined filters supported by the PhM editor. Only PhM filters are displayed: Composed (a composed filter can be specified), Entity class (excl) (keep all entity classes but remove all other components), Attribute
(excl) (remove all attributes), Generalisation (incl) (keep all entity classes and their generalisation hierarchies but remove all other components), Generalisation (excl) (remove all generalisation hierarchies), Cardinality (excl) (remove all cardinality constraints).

- **Search** provides the following options: PrM (search components in PrM specifications), Rules (search components in rule specifications), PhM (search components in PhM specifications). For each kind of search one has further options to show all links or a selection of links, e.g., according to a particular component or type of objects.

- **Workspace** provides the following options: Open (a particular workspace), Show (provides an overview of workspaces), Assign (a specification to a workspace), Move (a specification from one workspace to another one) and Remove (a specification from a workspace).

In Figure 9.16 a PhM specification has been retrieved (the result of selecting Open in the File menu). By selecting the Attribute (excl) in the Filter menu a viewspec is automatically created where all the attributes are removed (Figure 9.17).

![Figure 9.15: Functions to support complexity reduction in the PhM editor.](image)

**Services requested by the PrM editor** The PrM editor provides the same functions as the PhM editor but the filters are different. In Figure 9.18 a PrM specification has been retrieved (the result of selecting Open in the File menu).
The **Filter** menu shows only the pre-defined filters supported by the PrM editor: Composed (a composed filter can be specified), Port (total) (remove all ports), Port (partial) (replace all ports with abstracted ports), Flow (collapse) (collapse flows), Control flow (incl) (keep all control flows but remove all other flows), External agent (excl) (remove all external agents), Store (excl) (remove all stores) Timer (excl) (remove all timers) and Hierarchy (show a process hierarchy).

By selecting Composed in the **Filter** menu, the user then gets a list of possible filter to concatenate (Figure 9.18). She then specifies that a partial port filter (Port (partial)) should be followed by an inclusive control flow filter (Control flow (incl))
by ticking the empty boxes of the filters shown in a separate window. The resulting viewspec is shown in Figure 9.19.

Figure 9.18: Specification of a composed filter in the PrM editor.

Figure 9.19: A viewspec where ports are replaced by abstracted ports and all non-triggering flows are removed.

Services requested by the ERL editor Figure 9.20 shows the result of retrieving a simple rule specification in the ERL editor. To provide information about related components in the other models the user can either select the Search menu or select the Workspace menu. If the former is chosen the user can choose whether she wants to get a list of relevant components or if she wants to actually invoke the respective editor to show the diagrams that contain the relevant components.
This may be shown in separate windows. Let us say the user selects Open in the **Workspace** menu to get an overview of specifications in the current workspace (Figure 9.20). By clicking in the window the user can select what specification to see. By choosing the \( E1.0\{1\}\{1\}.1 \), the system displays a new window with the particular PhM specification (Workspace window), i.e., it shows the PhM viewspec in Figure 9.16. The user could then navigate to other specifications in the workspace in a similar manner, e.g., to the PID viewspec in Figure 9.19 by selecting \( P1.0\{1\}\{1\}.1 \) in the Workspace window, etc.

![ERL Editor with Workspace](image)

**Figure 9.20:** An ERL specification shown together with an overview of the current workspace and a context related viewspec.

A graphical tool for showing structural relationships between rules is available by choosing the **Relations** in the **Filter** menu. Subsets of rules can then be chosen by selecting **Subsets** in the same menu.

**Services requested by other modules** The other modules such as the explanation facility and the validation module will use subsets of the functionality requested by the editors. We will not go into detail about these but rather briefly describe how viewspecs are used by the explanation facility. The filtering mechanisms described above can be used to support explanation generation of specifications or parts of specifications. It is used as a supplement to explanation generation by visualising the relevant parts of a specification. The explanation generation facility may request a service from the viewing facility interactively or through a routine call. The latter is the case for viewspecs that can be automatically produced by the viewing facility, corresponding to simple viewspecs and composed viewspecs that are generated by concatenations of simple filters. A more detailed description of the use of viewspecs in the explanation facility is found in [154].
9.5 Chapter Summary

This chapter has briefly presented the PPP approach and suggested how the complexity reducing approach suggested in Chapters 5 to 8 can be exploited in the PPP CASE tool. This includes descriptions of how the PPP approach can be extended to deal with rules as well as complexity in the modelling process. We have discussed alternative solutions for incorporating rules in the PPP approach and also presented a prototype which was developed to show the feasibility of the coupling between process based and rule based approaches. The structuring and filtering mechanisms for TEMPORA languages only need minor modification in order to be used with the PPP languages and we have outlined how this can be done.

To extend the PPP architecture with facilities to support rule modelling and to provide complexity reducing capabilities, we suggested that three modules are added to the architecture: a rule editor, a viewing component and a specification integration component. The rule editor assists the developer in the rule modelling by providing conventional editing features, checking facilities, etc. However, the rule editor as such is not elaborated in this work. The viewing component allows the specification repository to be viewed from different perspectives. It also provides support for the generation of appropriate viewspecs for specification construction, validation and explanation generation. Other features such as support of manipulation of layout may also be added but are not provided in this work. Specification integration is only dealt with to a limited degree in the previous chapter and hence, specification of tool support is not provided. We have however detailed the major functionality of the extended PPP CASE tool as well as how its repository structure will be affected by the inclusion of a rule editor and a viewing component. It is worth noting that the incorporation of rules and complexity reducing features are possible without any major changes to the PPP tool environment. Implementation of the viewing facility
Chapter 10

Evaluation of the Complexity Reduction Approach

This chapter evaluates the suggested approach.

Sections 10.1 and 10.2 compare the results of the work in this thesis with the requirements presented in Chapter 4 and with the state-of-the-art approaches presented in Chapter 2, respectively. Section 10.3 reports some experiences from case studies. This is followed by a discussion of the general applicability of the suggested approach in Section 10.4.

10.1 Evaluation with Respect to Stated Requirements

This section compares the results of the work in this thesis with the requirements to an approach based on complexity reduction identified in Chapter 4.

To deal with fragmentation and information overload in specifications, requires that a sufficient set of the complexity reducing techniques (i.e., information hiding, structuring, viewing, translation and layout modification) described in Chapter 2 are provided. Since we have addressed how complexity reduction can be applied for both the PPP and TEMPORA languages, we will evaluate the work reported with respect to both. Table 10.1 provides an overview of the complexity reducing features of these languages.

In our work, complexity reducing facilities are embedded both as properties of the language and as properties of the methods and tools. The latter has been emphasised. Rather than extending the languages by defining new constructs to deal with more aspects of complexity reduction, e.g., information hiding, we have chosen to focus on the existing languages and provide additional support to improve their ability


<table>
<thead>
<tr>
<th>Modelling approach</th>
<th>Structuring</th>
<th>Viewing</th>
<th>Translation</th>
<th>Additional support</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPP</td>
<td>Inter-language links&lt;br&gt;Rule relations&lt;br&gt;Decomposition&lt;br&gt;Generalisation&lt;br&gt;Aggregation&lt;br&gt;Association</td>
<td>Filtering facilities</td>
<td>Explanation generation&lt;br&gt;Code generation</td>
<td>Updating and versioning of viewspecs&lt;br&gt;Complexity reducing approach</td>
</tr>
<tr>
<td>TEMPORA</td>
<td>Inter-language links&lt;br&gt;Rule relations&lt;br&gt;Decomposition&lt;br&gt;Generalisation&lt;br&gt;Aggregation</td>
<td>Filtering facilities</td>
<td></td>
<td>Updating and versioning of viewspecs&lt;br&gt;Complexity reducing approach</td>
</tr>
</tbody>
</table>

Table 10.1: Complexity reducing features in PPP and TEMPORA approaches.

to deal with development of large systems. Beyond structuring, the conceptual basis of the languages has not been changed with respect to complexity reducing features. Nor have we looked into how each concept and corresponding external representation of the languages could have been improved to satisfy the language dependent requirements, e.g., changing the graphical representation of symbols. This has been done in [117], and is not repeated here.

Evaluation with Respect to Language Dependent Requirements

The work that has focused on providing an integrated modelling approach to information systems based on rules has some impact on the conceptual basis as well as the external representation of the PPP and TEMPORA languages. This includes the coupling between the process based and rule based approaches in Chapter 5 as well as the inter-language links provided in Chapter 6. We therefore evaluate what is done on this matter with respect to the language dependent requirements in Chapter 4. The other structuring mechanisms are entirely embedded as properties of methods and tools, and thus dealt with in the next subsection.

Perceptibility The conceptual basis of the PPP and TEMPORA languages are slightly changed by the coupling between the process based and rule based approaches. The ERL remains the same but as we will see below, the PID\(^1\) has its semantics changed by this coupling. We believe that the perceptibility of the PID and ERL has increased by formally defining how they relate to each other. It is not unclear what can be expressed by the different languages and for those parts which

\(^1\)The concepts of buffers and resources are not addressed in this thesis. Hence, the PrM also has its semantics changed since PID and PrM are basically the same languages, only their external representation differs.
are overlapping, it is stated what constructs in one language that have a counterpart in the other language and what they are. In general, the inter-language links increase the perceptibility of the TEMPORA languages by stating how they relate to each other.

Similarly to the increase of the perceptibility of the conceptual basis of the PPP and TEMPORA languages there is certainly also an increase of the perceptibility of the external representation. In particular, the visualisation of parts of rules, e.g., explicit representation of control structures, contributes to increased understanding of complex rule structures and relationships between rules. This has also been confirmed in case studies [142], as will be elaborated in Section 10.3.

**Expressive power**  The expressive power of the PPP approach has increased by the incorporation of rules as suggested in Chapter 9. We believe that this mends some of the deficiencies of PPP, e.g., rules are not any longer just pieces of informal text associated with PPP specifications. We will now be able to deal with rules in a systematical manner and we also allow rules to be expressed at different abstraction levels. Thus, the PPP languages has become better suited for modelling of systems which are characterised by large collections of rules, e.g., task processing systems.

As far as the TEMPORA languages are concerned, only the expressiveness of the PID has been influenced by our work. The coupling between the ERL and the PID gives the PID (and thus PrM in PPP) a precise logical and executional semantics. It should however be noted that the notion of buffers such as found in [93], is not supported in this approach. This has the consequence that if a process requires that more than one trigger arrives, these must arrive at the same time to start the execution of the process. Otherwise the execution of the process will fail. To deal with triggers that do not arrive simultaneously at a process, the executional semantics of the PID and ERL need further investigation. This is being addressed by a current report by P. McBrien [84].

**Expressive economy**  The expressive economy of the conceptual basis remains the same since no changes to any of the PPP and TEMPORA languages have taken place. The expressive economy of the external representation has however, increased by the fact that the visual features of the PID and the ERT are exploited for the rule language through the coupling to those languages.

**Method and tool potential**  The method and tool potential of PPP and TEMPORA languages have certainly increased by providing a formal definition of the semantics of the PID and the inter-language links. The structuring mechanisms lend themselves to comprehensive computerised support. The links facilitate advanced searching and navigation in specifications, and consistency checking across specification boundaries. The PID and the ERL are overlapping, and the coupling between these languages can also be used to ease rule conceptualisation. This is shown in
Chapter 8 by generating rule skeletons from process descriptions represented in the PID. Again the benefits of a visual representation are exploited.

As for the tool potential, the method potential has increased by providing the basis for a deeper understanding of how the languages relate to each other. For instance, it is easier to develop comprehensive guidelines for how to use the languages in practical modelling when it is clearly stated what their constructs mean and how they relate.

**Reducibility** The PPP and TEMPORA languages' abilities to deal with large and complex systems have increased by improving their structuring features. This is achieved as follows: First, the incorporation of rules in PPP together with structuring mechanisms contribute to reduce the gap between specifications at different abstraction levels. The integration of rules with the PhM and the PrM contributes to bridge the gap between declarative specifications at higher abstraction levels and algorithmic specifications at lower abstraction levels. The developer can choose between PLD and ERL to specify the internals of processes. In addition, she can use the links to express relationships between components of other specifications at the same or at different abstraction levels.

The suggested approach allows rules to be explicitly expressed at increasing level of detail, and thereby it reduces the complexity of the modelling of systems that contain many rules such as task processing systems. Rules are not just pieces of text placed in a more or less *ad hoc* manner in the specifications and, e.g., a high-level business concept is not enforced to be expressed in a concrete implementation-oriented representation too early in the development process.

Second, the coupling between the process based and rule based approaches can be used as a basis for providing an overall structure of rules. Thus, the PID/PrM can be considered as a grouping facility for rules. This has also been adopted successfully by another approach, COMEX [149] to model and structure rules in knowledge based systems.

Third, we believe that the coupling between the process based and rule based languages together with the other inter-language links provide the necessary links to mend the major problems of fragmented specifications *within* each specification level both in TEMPORA and PPP. These links are implicit and thus, no constructs are provided in the language. The external representation of the links is built into the editors supporting the languages (see below). As far as relationships between various abstraction levels are concerned, we have *not* provided additional concepts in the languages for this. The reason is that we feel the languages already are comprehensive enough and that we want to keep the number of concepts at a reasonable level to maintain the perceptibility of the languages. Thus, inter-level links are provided as properties of the methods and tools.

The other abstraction facilities in the languages such as hierarchical data abstrac-
tions and decomposition remain unchanged, c.f., Chapter 3. It should however be mentioned that the concept of complex entity class in the ERT needs further clarification. Currently, it is only a grouping facility and such an object has no specific properties beyond the properties of the components from which it is built.

Evaluation with Respect to Method and Tool Dependent Requirements

We have suggested a complexity reducing approach for the PPP and TEMPORA languages. We have developed facilities to view specifications of any size from different (and optionally related) perspectives as well as facilities to enter information into large specifications in a flexible manner. We have already evaluated above those parts of the approach which are connected to the languages. We will now examine the other parts of the suggested approach with respect to the method and tool dependent requirements in Chapter 4.

Reducibility In this work, the emphasis has been put on structuring and viewing. Links are provided to relate information at different levels. As mentioned above, the structuring mechanisms provide the coupling between components which are located within the same specification and between components which are located in different specifications. By using the structuring mechanisms suggested in Chapter 6 (with the necessary adaptations as described in in Chapter 9 for the PPP), it is possible to express explicitly how rules relate to each other at different levels of abstraction and to the components expressed using the ERT/PhM and the PID/PrM.

The inter-rule links allow rules at different abstraction levels to be explicitly related. The following inter-rule links were suggested: causes, uses, has_info_derived_by, constrained_by, is-related-to, suspends and overrules. All the relations except from the three latter can be derived from a TEMPORA (PPP) specification expressed using the ERT, the ERL and the PID (the PhM, the ERL/PLD, the PrM). The tree latter relations must be manually entered and maintained. The inverse of these relations can always be derived. As with the rule relations we have also strived for defining a minimum set of inter-language links and then used these to derive other relevant links. Thus, all the relations lend themselves to computerised support, as is briefly outlined below. What remains to be done is to further investigate in case studies the suggested set of links in order to attain a more complete set of structuring mechanisms.

The structuring mechanisms that are implemented in the TEMPORA capture tool by SISU\textsuperscript{2} are:

- Intra-language links which include motivates and necessitates. Due to limited

\textsuperscript{2}Swedish Institute for Systems Development, Stockholm.
project resources a visual tool for showing the rule relationships has not been implemented.

- Inter-language links ERL, ERT and PID. This also include an associates relation between rules and the ERT. This links must be manually entered and maintained.

Entering of each structural link takes place in the various editors. The details of the implementation of the links in TEMPORA and how the links are entered is described for each editor in [139]. Inter-level links except from links between rules at different abstraction levels, were not dealt with to the same extent in this thesis since there is already available work that address this such as the ONE-R algebra [157] and versioning of specifications at different abstraction levels [3]. Due to limited project resources in TEMPORA such inter-level links were not explicitly supported, so the specification database in TEMPORA was simply copied from one level to the next and then further elaborated.

Thus, a subset of the structuring mechanisms suggested in Chapter 6 are already implemented in TEMPORA. In PPP, we have developed the basis for including all the links and in the new version of PPP under development these will be implemented. We have already specified in Chapter 9 the major changes/extensions that is required to the PPP repository structure in order to include the links. It also shows that such links can be provided without dramatic changes to the PPP repository structure. The same holds for the inclusion of rules in the PPP.

The filtering mechanisms allow a set of specifications to be viewed from a particular perspective by focusing on relevant issues and suppressing irrelevant details (viewing). The suggested approach seeks to meet the wide range of needs by developers and end-users by allowing flexible creation and use of viewspecs. By defining a set of filters for the PPP and TEMPORA languages, specifications expressed using these languages can be examined from different and optionally related perspectives.

Our approach to creating filters gives a high degree of flexibility. It allows the user to use pre-defined filters and also define her own filters. In addition, such filters can be concatenated to form composite filters. As far as the pre-defined filters for the TEMPORA and PPP languages are concerned, we believe that we have defined a sufficient set (experiences are described below). The pre-defined filters can generate superficial viewspecs (e.g., the overview of the processes at different levels of decomposition) and narrow viewspecs (e.g., the component viewspec), as well as combination of such viewspecs (e.g., providing details for certain components in a diagram and leave out others). Only a subset of the filters which were specified in Chapter 7 are implemented in the TEMPORA capture tools (Figure 10.1):

- **Value class filters** are implemented by the hide value class and show value class commands in the Layout menu.

- **Cardinality constraints filters** are implemented by the Hide Mapping and Show Mapping commands in the Layout menu.
10.1. Evaluation with Respect to Stated Requirements

Figure 10.1: Complexity reducing features in the TEMPORA capture tool.

- **ERT component filters** are implemented by the Mark to Hide/Unmark, Hide all Marked and Show Hidden commands in the Layout menu. Each symbol to be shown hidden must be explicitly marked, that is, also independent objects such as relationships between entities must be marked. It is also possible to select an entity and specify how many surrounding objects (i.e., number of transitive relations) that one wants to look at. This is done by the Neighbourhood command in the Object menu.

- **PID component filters** are implemented by the Mark to Hide/Unmark, Hide all Marked and Show Hidden commands in the layout menu. In a similar manner as for ERT component filters, it is possible to specify the number of transitive relations (Neighbourhood command in the Object menu). Also here independent objects such as flows must be marked to be removed.

However, as far as filtering across specification boundaries is concerned, we have only specified and outlined with examples how this can be done. To what extent this should be accompanied by a set of well-defined filters and thus form the basis for automated filters, is yet to be investigated. This requires that the existing filters are tried out and also that filters that are frequently used or time consuming to create are identified.

Moreover, the suggested approach allows multiple viewspecs to co-exist as well as updating of viewspecs. By allowing viewspecs to be updated, the developers can work with specifications that contain the relevant issues whilst not being disturbed by irrelevant details. Various views of specifications may be useful at different stages and the approach gives the developers support in managing full specifications and associated viewspecs through its interplay with a versioning system. In this work, the versioning system developed in [3] has been revised to fit better with an approach
based on complexity reduction (see below). Flexible presentation of viewspecs is also provided by supporting both non-persistent and persistent viewing. It is possible to selectively present fragments (viewspecs) of the specifications (in graphical, tabular and textual form), directed by menu commands for documentation/report generation. This is also important in order to deal with issues such as space constraints of the medium, e.g., screen and paper size.

Multiple viewspecs, updating of viewspecs and versioning are not implemented in the TEMPORA capture tools. Both non-persistent and persistent viewing are supported although only to a limited degree. The reasons are that only some of the filters are implemented and the result of a filter is not an independent viewspec, i.e., it is not independent of the originating specification. The result of performing a filter is a new picture of the repository and is connected to all other pictures in the given model (A graphical model in TEMPORA consists of a set of pictures). For instance, a new picture generated by the Neighbourhood function contains components with the same identifiers as the components in the diagram from which it is generated. They are thus considered as the same components by the system (which appear in different pictures). A change performed on this picture will also be reflected in the repository and consequently, it can be accessed in other pictures. As a result the implementation of these features in the TEMPORA capture tool deviates from the approach to support the building of specifications based on independent viewspecs as suggested in Chapter 8. In spite of this it gave us useful input to how the process of developing specifications based on complexity reduction should be carried out. The implementation of the new version of PPP will include the complexity reducing features. It follows the approach as described in Chapter 8. In this work we have specified how such features can be included in PPP by detailing the impact on the PPP repository structure and the functionality of the tool (Chapter 9).

Translation is only addressed to a limited degree by the coupling between the PID and the ERL. These languages are overlapping and rule skeletons can be automatically generated from PID specifications. However, translation is addressed in two companion theses [73, 46] and is beyond the scope of this text. Layout modification has also only been briefly addressed, and is a weak point of our work. To provide comprehensive tool support for the complexity reducing approach implies that modification of layout is supported. However, it will be a major effort to provide sufficient support for, e.g., redrawing of large diagrams, and we have limited ourselves to outline relevant work in this area.

Combinations of the above techniques have been suggested to improve presentation of specifications. Viewing and modification of layout are often used in the filtering process. The same applies to structuring and viewing. We have also combined viewing, structuring and translation to ease rule conceptualisation. In general, the various complexity reducing techniques suggested in this thesis can all be combined with other techniques. However, the flexibility of the tool environment determines to what extent the different techniques can be combined, e.g., a sufficient set of structuring mechanisms must be implemented in order to generate viewspecs across specification boundaries.
Changeability has been addressed by providing structuring mechanisms and allowing new specification details to be entered into viewspecs as well as full specifications. The structuring mechanisms facilitate advanced searching and navigation between specification components. Thus, it is possible to localise place of change and also find related information in case changes must be propagated to other specifications. It is possible to search and/or navigate between specification components by using the diagrammatic as well as the non-diagrammatic components. This also facilitates easy access from one specification to related components contained in other specifications.

Our approach gives the developer the possibility of working with specifications at a size which may be suitable for the task to be undertaken. The developer can choose appropriate granularity of work modules according to the task to be carried out. Rather than operating on a full specification, relevant viewspecs can be applied at different stages of the development process (e.g., in rule capture) and then eventually integrated into a new version of the full specification. It is fully up to the developer to specify the granularity of the work modules\(^3\), that is, it is a user decision to specify granularity. Viewspecs can be directly updated and revisions of viewspecs are managed by the versioning system. The process of building up a new specification based on a set of updated viewspecs needs more computer-aided support. The synthesis of viewspecs and other specifications is only partly addressed, and further work is necessary.

A weak point as far as changeability is concerned is the support for updates of aggregates. It is limited to support of inspection. However, updates of viewspecs imply that atomic units as well as aggregates may be updated. The suggested approach deal with updates of atomic units but as far as aggregates are concerned the support is limited. The basic problem with such composite objects is that one does not know which of the components which make up the composite object, are influenced by the update. An example is an update of a complex object. In our approach, the support is limited to show which components that are related. The developer may then inspect each to determine which components that should be updated.

The versioning and configuration control system [3] is used to keep track of various specifications and various versions of specifications. However, certain modifications to the versioning system is suggested in Chapter 8 to support an approach based on complexity reduction. The suggested facilities for managing viewspecs in the modelling process fit well together with the solutions to versioning as proposed in [3], and can easily be integrated into the current implementation, as shown in Chapter 9.

Modelling freedom We now outline how the various means suggested in this thesis, will impact on the way of working:

\(^3\)The granularity must be equal or courser than the finest granularity of the versioning system.
- **Finer granularity of work modules.** The developer can choose the granularity of a specification by producing an appropriate viewspec containing the relevant details. By using viewspecs as a basis for specification building the manipulation of large diagrams can be reduced.

- **Flexibility.** Multiple perspectives of specifications are allowed and also conflicting viewspecs can be explicitly modelled. This offers the developers an opportunity to make conflicts visible in the modelling process and also provides a basis for, e.g., collaborative conflict resolution.

- **Possible to work with incomplete specifications.** This includes the ability to enter specifications in an incremental and fragmented fashion, i.e., starting with a “skeleton” of a specification structure and then adding details whenever appropriate. This is achieved by allowing the developer to apply viewspecs at any stage to support the modelling process.

- **Postpone procedural specification as much as possible.** The possibility for choosing between a declarative and a procedural specification of the processing parts of a system by incorporating rules in PPP, gives the developer greater flexibility in the modelling process. For instance, this allows her to postpone any detailed descriptions of “how” aspects till later stages of the development.

### 10.2 Evaluation with Respect to Other Approaches

This section evaluates the suggested approach with respect to some of the modelling approaches described in Chapter 2. The complexity reducing features of the modelling languages in Chapter 2 are already summarised in Tables 2.1 and 2.2. Table 10.1 provided a similar overview of the complexity reducing features of the PPP and TEMPORA languages. These tables form the basis for our evaluation.

A commonly used evaluation strategy is to give weights to the columns of the table. This is done in Table 10.2 where we evaluate the modelling approaches’ complexity reducing features. The weights indicate the relative importance of the features mentioned in the rows in Tables 2.1, 2.2 and 10.1.

The complexity reducing features in the table are classified as **, * and - depending on the degree to which the modelling approach supports them. The score ** means that the complexity reducing feature is well covered by the approach, * means that it is medium covered and - means that only little support is provided or the issue is not addressed at all. We will limit ourselves to describe the evaluation for some of the approaches to show how we have used the scores **, *, and -. The details of the approaches’ complexity reducing features are already described in Chapter 2 and therefore we do not repeat these descriptions.

As we can read from Table 10.2 most of the old languages such as the original ER language and classical Petri nets are poor as far as complexity reducing features are
### 10.2. Evaluation with Respect to Other Approaches

<table>
<thead>
<tr>
<th>Modelling approach</th>
<th>Information hiding</th>
<th>Structuring</th>
<th>Viewing</th>
<th>Translation</th>
<th>Additional support</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ERC+</td>
<td>-</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DFD</td>
<td>-</td>
<td>**</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CAPS</td>
<td>-</td>
<td>**</td>
<td>**</td>
<td>-</td>
<td>*</td>
</tr>
<tr>
<td>OOA</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Wink-Brock's</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OMT</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OSA</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>XCON</td>
<td>-</td>
<td>**</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SAPIENS</td>
<td>*</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>COMEX</td>
<td>-</td>
<td>**</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Viewpoint Resolution</td>
<td>-</td>
<td>-</td>
<td>*</td>
<td>-</td>
<td>**</td>
</tr>
<tr>
<td>BNM</td>
<td>-</td>
<td>*</td>
<td>*</td>
<td>-</td>
<td>**</td>
</tr>
<tr>
<td>Statecharts</td>
<td>-</td>
<td>**</td>
<td>**</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ERAE</td>
<td>-</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GIST</td>
<td>-</td>
<td>*</td>
<td>*</td>
<td>**</td>
<td>-</td>
</tr>
<tr>
<td>ARIES</td>
<td>-</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>-</td>
</tr>
<tr>
<td>Abstraction-based sw. dev. approach</td>
<td>*</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>**</td>
</tr>
<tr>
<td>ViewPoint or. dev.</td>
<td>-</td>
<td>-</td>
<td>*</td>
<td>-</td>
<td>**</td>
</tr>
<tr>
<td>HICONS</td>
<td>-</td>
<td>**</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>KA approaches</td>
<td>-</td>
<td>-</td>
<td>**</td>
<td>-</td>
<td>**</td>
</tr>
<tr>
<td>Flowcharts</td>
<td>-</td>
<td>-</td>
<td>**</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PPP</td>
<td>-</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>TEMPORA</td>
<td>-</td>
<td>**</td>
<td>**</td>
<td>-</td>
<td>**</td>
</tr>
</tbody>
</table>

Table 10.2: *Evaluation of complexity reducing features in modelling approaches.*

Concerned. They are both totally flat and no facilities are provided to deal with large systems. Thus, all their complexity reducing features receive the score - . ERC+ receives the score * on structuring feature because it supports the classical hierarchical abstractions. These constructs allow information to be expressed at increasing level of detail and also provide a means to structure related components *within a diagram*. However, it provides *limited means to split up a model* into more manageable units and the resulting specifications are *flat*. This is common for most data-oriented languages although some provide concepts to build more complex constructs from a set of basic constructs. As we can read from the table some of the more modern languages such as Statecharts and the object-oriented languages receive the score ** on their structuring features. For instance, Statecharts which are an extension of the traditional state transition diagrams provides a number of mechanisms to deal with large specifications such as hierarchical abstractions, zooming facilities and unclustering. Thus, they have extensive support for abstraction, structuring and viewing.

As we can see from Table 10.2, there are no approaches that focus on all aspects but they rather concentrate on some of the complexity reducing techniques. We consider
the PPP and TEMPORA approaches as being strong on structuring and viewing features, and they also accommodate a way of developing specifications based on viewspecs. In addition, the PPP provides extensive support for translation of PPP specifications into executable specifications and also provide explanation generation facilities for executable as well as conceptual specifications [46, 153]. The translation features are not a contribution of this work and thus, left out in the remainder of this text. As far as the other features are concerned we will focus on those which are supported by PPP/TEMPORA and are contributions of the work in this thesis, and compare these to relevant features in other approaches.

The basic structuring mechanisms supported by the PPP and TEMPORA approaches are similar to those found in approaches based on the same languages, e.g., classical hierarchical abstractions in ER based languages and decomposition in DFD based languages\textsuperscript{4}. However, to allow rule specification at different levels of abstraction and to provide links between rules at different levels of abstractions as well as between rules and the ERT and the PID, are novel to the PPP and TEMPORA approaches. In Sapiens rules are attached to data definitions and stored as objects. Thus, information hiding and encapsulation are the most important complexity reducing facilities whereas in PPP and TEMPORA the ERT and the PID are used as grouping mechanisms for rules together with explicit relations between rules. The use of the PID as a structuring mechanism for rules has also been adopted by COMEX [86, 149] to manage large collections of rules for knowledge based systems. XCON takes yet another approach to rule structuring by using control techniques and the use of subgroup schemas which provide abstractions describing sets of rules that allow developers to index into the rule base.

Visualisation of control structures of rules has been supported in Nexpert [87] which is an expert system shell developed by Neuron Data Inc. The control structures correspond to the causes relation in our approach. However, we provide the foundation for a more flexible visualisation of rules through the more comprehensive set of rule relations defined in Chapter 6. All the relations can be derived except from the relates-to overrules and suspends which must be manually entered.

Another structuring mechanism which is novel to our approach is the coupling between the rule based and the process based approaches. This is not to say that other hybrid approaches do not provide comprehensive structural links between their dynamic languages, e.g., OMT [110] and OSA [32] provide comprehensive structural links between their flow oriented and state oriented languages. However, to our knowledge there are no other approaches that provide a formal definition of the coupling between a DFD based language and a logic based language including temporal aspects as provided in our approach.

Mechanisms to allow definition of different views are well known means to abstract away irrelevant details both in specification languages for static modelling (e.g., the concept of scenario for defining different views of a domain in the PhM [121]) as well

\textsuperscript{4}Since this has not been the focus of our work we will not elaborate upon this in this text. We refer the interested reader to [117] for a more detailed description.
as for database applications (e.g., SQL views [25]). It is however not until recently the viewing feature has been widely adopted by other approaches (beyond static modelling). CAP is an approach based on Data Flow Diagrams augmented with multiple views where a distinction is made between facilities for providing overview pictures (summary views) and facilities for focusing on certain details (navigation structures and focused slices) [77]. Moreover, object-oriented approaches provide a wide range of viewing facilities. The notion of layers in OOA [24] gives the developer the possibility to choose what concept to be displayed at a time. OSA [32] introduces the notion of views/high-level object classes, where a distinction is made between exploded and implored views. Higher level abstractions are used to represent fundamental system concepts (imploded views), whereas lower level abstraction unfold supporting detail (exploded views). Thus, the basic idea of generating multiple viewspecs in our approach is quite similar to the way of generating view in these approaches. Where our approach to filtering differs from the other approaches is in the scope and flexibility of viewspec generation. We define a wide range of different filters and if none of these are suitable, the user may define her own or even use a combination of manual and automated filters. Other differences include updating of viewspecs and versioning of viewspecs as are further elaborated upon below.

Some CASE tools today have built in certain complexity reducing features in their tools but these have very little formal basis (mainly functions implemented in the tools). One example is the grouping function provided by the IEF CASE tool where entities and relationships can be grouped to form higher level objects. However, such an object has no own behaviour and can only be considered as a vague composition facility. Another example is the concept of scenario in RDD (Requirements Driven Development [2]) which supports the generation of different views of a textual specification and allows multiple views to co-exist. The limitation of this approach is the lack of formal specifications.

Moreover, the concept of filters is supported by The Bachman Tool Set. However, its filtering functions are rather limited and only non-persistent viewing is supported, cf., Chapter 2. In spite of this, the filtering feature of The Bachman Tool Set gets an almost top score as far as scope of views (usability feature) is concerned in a recent evaluation of CASE tools [6]. We choose to interpret this as a sign that ways of dealing with complexity in ICASE tools are uncommon and filtering seems to be considered as a promising way of dealing with large amounts of detail. Although most of the approaches and their more advanced complexity reducing features (Table 10.2) are not included in contemporary commercial ICASE tools, some of their advanced features will most likely be found in the next generation of ICASE tools.

As we see from the above, each of the parts suggested in this work represent some novelties in their own but we certainly believe that the complexity reducing approach suggested in this thesis brought about some new ideas. As mentioned above viewing is one complexity reducing technique which has gained more popularity in recent years. This has also resulted in a number of approaches which are based on views or viewpoints. Our approach allows multiple perspectives to be adopted when developing information systems, and can thus be considered as a viewpoint-oriented approach. We will therefore compare the main characteristics of our approach with
respect to the viewpoint-oriented approaches Viewpoint Resolution [70], ViewPoint-Oriented Development [38] and ARIES [63] described in Chapter 2:

- **ARIES** [63] is an environment to experiment with support for analysts in modelling target domains and in entering and formalising system requirements. It allows information to be presented and entered in a variety of languages. Like our approach ARIES also stresses the importance of supporting different viewpoints. Where our approach differs from ARIES is that the different PPP/TEMPORA languages are to a large extent required to convey distinct information. ARIES however, has a single internal representation of its information and provides a set of mapping from/to its internal language to the desired external language used for presentation. Thus, a change in one viewpoint results in a change in the internal representation and consequently, other presentations that use the same information will be updated. This is in contrast to our notion of viewspecs which are only loosely coupled to other specifications. We also allow viewspecs to be updated even if an inconsistency among viewspecs or between the originating specification and the viewspec may result. Changes are not propagated automatically to other specifications. Hence we provide the developers with the abilities to model conflicts explicitly, to explore alternative modelling solutions, to generate viewspecs tailored to the skills, background, etc. of the actors involved, etc.. The key point in solving the problems of inconsistency is to relate a viewspec to a consistent system specification. We believe that the inclusion of the versioning mechanism as developed in [3] with the necessary modifications/extensions as suggested in this work provide the necessary support to manage such viewspecs in the development process. To our knowledge the use of viewspecs as independent work modules which can be updated and are associated with a consistent version of their originating full specifications is novel to our approach. It should however, be mentioned that current implementation in the TEMPORA capture tool takes the ARIES approach as far as propagation of updates of viewspecs are concerned.

- **Viewpoint Resolution** [70] is developed to assist validation of rule specifications. Rules are grouped together in viewpoints and Viewpoint Resolution provides a way of detect and resolve conflicts between such viewpoints. Thus, it addresses the fourth activity in our complexity reducing approach, namely Inclusion of changes which has only been briefly addressed in our work. Viewpoint Resolution is complementary to our work in that it may provide useful input for us in future development of integration of viewspecs that are expressed in ERL.

- **ViewPoint-Oriented Development** [38] is not tailored to any particular modelling languages but their work include an organisational framework developed to encompass multiple viewpoints (called ViewPoints) and various kinds of support to develop systems based on such viewpoints. ViewPoint-Oriented Development is also complementary to our approach in that it addresses more the overall organisational aspects of viewpoint-oriented development which we only address to a limited degree, e.g., different roles of software development participants. Similarly to Viewpoint Resolution, ViewPoint-Oriented Development
Development also provides support for consistency checking between pairs of ViewPoints. As mentioned above, this is not addressed in our work.

However, where ViewPoint-Oriented Development aims at developing a general framework for systems development based on viewpoints which are independent of the modelling languages that are used, we have developed an approach which apply viewspecs to assist modelling for large and complex specifications expressed in the PPP or the TEMPORA languages. In addition, we establish a common ground for viewspecs by requiring that a viewspec is always associated with a full specification. Our approach is developed to assist systems development approaches in dealing with large and complex systems, where we have based our work on well known Information Engineering languages and principles. Thus, complexity reducing approach is used together with the PPP and TEMPORA approaches to assist the major modelling processes, that is, elicitation, conceptualisation and validation.

Our work in seeking new ways of reducing complexity in specifications has mainly been achieved by elaborating on the idea of abstraction. One of our main aims has been to allow all relevant details to be included in a specification and at the same time make it comprehensible for the actors that participate in the modelling process. A striking feature however is that the idea of abstraction as a specification paradigm is not widespread. To our knowledge, the only abstraction-based software development approach is developed by V. Berzins et al. [12, 77]. The key idea being to provide mechanisms called black box descriptions at different levels of abstraction which enable description of external behaviour of any system (and sub-systems) to be distinguished from the internal mechanisms that eventually are used to realise that behaviour.

The above evaluation contains some subjectivity. A closer study of the different approaches (e.g., formal evaluation and also a broader selection of approaches) is required to make more detailed statements about the different approaches’ abilities to model large and complex systems. However, this study indicates how the major features of how the work suggested in this thesis relate to contemporary approaches’ abilities to deal with large and complex systems.

10.3 Experiences using the Suggested Approach

This section reports some experiences from applying complexity reducing facilities on examples and case studies. The suggested approach has been tested as follows:

- Case studies in TEMPORA.
- A case study done by project students [68].
As already mentioned above, due to time constraints only limited tool support has been available for testing out the suggested approach. Thus, those parts which where not supported by tools have been developed manually.

The tool support in TEMPORA that has been used in case studies was developed by SISU during the TEMPORA project. The implementation of the new revised PPP CASE tool is however, in an immature state so any experiences using the tools for modelling cannot be reported.

Case studies in TEMPORA

Background During the TEMPORA project (1989-93/94) several case studies were conducted as a part of the project or as separate work conducted by the TEMPORA partners, e.g., the Oil Processing and Library case studies described in Chapter 1. Here we will limit ourselves to the description of experiences during the work on the Sweden Post case study since these cover the relevant issues as far as complexity reduction is concerned.

The case study was started at Sweden Post in the third year of the TEMPORA project [133]. During the final two years of the project, this case study was updated both to encompass more domain knowledge and to make use of new features introduced in the TEMPORA languages. Also, for the first time the automated compilation of a runtime system based on the conceptual level specification could be made, and thus the case study could be fully tested on a runtime platform. The revised Sweden Post case study represented the first major testing of the integration of the conceptual and runtime levels in TEMPORA. With some 160 executable rules\(^5\), 128 ERT entity classes, 116 ERT relationships and 15 transactions; it represents a typical small to medium scale commercial application. Thus, it can be regarded as a true test of the three areas of the TEMPORA approach, namely: capturing conceptual models in the ERL, ERT and PID, designing and compiling an executable specification in TEQUEL from those models, and executing the TEQUEL program on a runtime system built on top of the SYBASE RDBMS. Only the former is addressed here. The interested reader is referred to [139] for a more complete description of experiences using the TEMPORA approach in the Post case study.

We limit ourselves to the experiences in developing the extended version. At that time the case study had reached a size where the specification complexity had become a serious problem for communication among developers as well as between developers and users, and contains therefore the most relevant aspects as far as complexity reduction is concerned. We distinguished the following major tasks where complexity reduction could be applied:

1. The overall understanding of the problem domain. How to get a good understanding of the domain and its documentation?

\(^5\)Also a large number of informal rules is captured in the system.
2. The maintenance of the initial version of the case study. How to get a good understanding of the already specified system and perform the necessary changes?

3. The details of the TEMPORA conceptual model. How should the different parts be expressed using the new features and modified version of the TEMPORA languages? In particular, how to deal with the constructs and how to express domain knowledge using these concepts?

4. The transformation from the conceptual model to a design model. What details need to be added to produce specifications that contain all the design details necessary to generate an executable system?

We will summarise the major experiences as follows: use of search and navigation facilities, rule capture using the PID and use filtering mechanisms.

**Search and navigation facilities** We have already mentioned above that the case study has reached a size where the specifications are large and complex. The fact that these specifications are strongly related implied that it was necessary for the modelling process to have flexible mechanisms to search the specifications to find information at any point in the process. Various search functions were used. Structuring rules according to ERT and PID are supported by the capture tool and were useful in the sense that it made it easy to see what rules belongs to an ERT construct or a process. By selecting an object in ERT or PID the corresponding rules can be edited one at a time or printed all together. Here the search facility works well and makes it possible to get an overview of what rules are affecting an ERT component (i.e. an entity, a relationship or a value) or a process.

The search can of course also be made on all rules, ERT or PID components where the result can be presented in a list. From this list it is possible to select one component to be investigated in detail by opening its form. One of the most useful functions of the search facility is that the search result can be combined with a new search. That is one component in the result can be selected and its neighbours can be listed. Then one component in the new list can be selected in turn, to be presented by its form, its picture(s) or its neighbours can be searched and so on. It is also very useful to be able to make the search both by selecting a component by clicking on it in a graph and by typing in a string which may be only a part of a label. Taken together this makes it quite easy to find the way through the specification. Navigation can be performed in a similar manner. Instead of using a set of objects as a basis for further searching one may open the specifications in which the objects are located.

**Rule capture using PID** PID turned out to be useful for assist rule capture in TEMPORA. There were at least three ways that the PID were used to support rule capture. First, PID diagrams were useful to give overview of the UoD in a user-friendly manner. Second, for each process one could identify the event-action rule(s) that described the behaviour of the process. Third, for each process one
could also identify the rules that governed the behaviour of the process, that is, the
rules that described the constraints of the process' behaviour. Rules associated with
other components of the PID specification such as ERT views and timers could also
be captured. Using the PID in this way was particularly useful in modelling parts
of the case study comprising many dynamic rules (e.g., handling of delivery notes).
It should be noted however, that the PID capture tool does not support the concept
of ports, something which is necessary for utilising the the full power of the PID for
rule capture (not implemented due to resource constraints).

The use of filtering mechanisms  When developing the case study we frequently
entered the situation that our specifications contained too much information and
thus, this confused the modelling process. As in all development of large and com-
plex systems we had to find ways to locate and focus at the relevant details. For
some parts of the case study (e.g., delivery note handling), we used the set of filters
suggested in Chapters 7 and 8 to assist in the modelling process. They turned out
to be useful as basis for understanding among developers as well as basis for gain-
ing insight to the domain for the individual developers. Although no support was
provided in the capture tools, the case showed that systematic use of filters were
useful to deal with complexity in the specifications. In particular, this applies to
rule capture by exploiting the visual features of the ERT and PID languages.

This way of abstracting away details was found very useful in TEMPORA when
specifying rules using the process model. Using different models in developing a
system implies that we look for certain features of the models at a particular time,
i.e., integration points between the models. Thus, it was useful to have filters which
are based on the coupling between different model, easening the problem of finding
the relevant parts of the specifications. Different filters of different specifications
(e.g., by having different windows on the screen) could be shown at a time. More
comprehensive tool support is needed, e.g., for the creation of viewspecs and inclusion
of changes into full specifications based on viewspecs. The visual features of the ERT
and PID languages are exploited to ease rule capture. Filters were used to generate
viewspecs containing the relevant details. The ERT viewspecs were used to ease
formulations of ERT access expressions and the PID viewspecs where used as a
basis for generating rule skeletons. More specifically, the former was useful when the
ERT expression contained several different entity classes (i.e., long navigation chains
in the ERT, where the entities could be located in different pictures), whereas the
latter was particularly useful when the number of flows into a process was high or
one or more flows contained many variables.

A Case Study Done by Project Students

Two project students tested out an intermediate version of the complexity reducing
approach\textsuperscript{6} in a case study of an office information system [68]. They updated a

\textsuperscript{6}In [68], it is referred to as an abstraction-based approach.
previous model of an office system for the Department of Computer Science and Telematics at NTH\textsuperscript{7}. The actors involved from the department were employees with comprehensive background in modelling. The project was supervised by the author.

During the case study, all the four activities of the complexity reducing approach were addressed: filtering, presentation, entering new specification details and inclusion of changes. They used viewspecs together with full specifications to validate their revised model. Furthermore, these viewspecs were also used as a basis for entering new specification details during the modelling sessions with the end-users. The major results of this experiment were the following:

- \textit{Filtering}. During the case study various PID filters were used to generate different viewspecs of the office system. The students felt that the set of PID filters were sufficient however some filters needed to be improved, e.g., the graphical notation for some of the viewspecs was confusing. Thus, the students also experimented with different graphical notations for viewspecs. This provided useful input for the author in the work reported in this thesis and most of the students' recommendations have already been taken into account in the filtering mechanisms suggested in Chapter 7 and will not be repeated here. The interested reader is again referred to [68] and to earlier reports on filtering mechanisms by the author [134, 142].

The process of generating viewspecs from a specification has been carried out manually. Generating different types of viewspecs using filters was a tedious task, and managing different versions of viewspecs was also difficult. The experiment made it obvious that the process of creating appropriate filters must be accompanied by comprehensive computerised support.

- \textit{Presentation}. The full specifications together with associated viewspecs were used as a basis for communicating with the employees in the department. Complexity reducing techniques were also combined to improve the readability of the viewspecs. In [68], the following statements are made about the appropriateness of these in the validation process: 'For us, the new diagrams made the communication with the employees easier. One general comment was that the diagrams with abstractions were much easier to understand than the diagrams in [76], ... The employees did not need much time to understand the diagrams when we presented them according to the description in [142]. They could concentrate on the process they really were going to validate, and go back and forth between diagrams to get an overview or detailed information.'.

- \textit{Entering new specification details}. As far as entering of new specification details into viewspecs is concerned, the following observation was made [68]: 'It was also practical with diagrams that did not contain too much information when the employees wanted to do some changes, e.g., put on a new flow or port structure.' Thus, it was considered less time consuming to use viewspecs as a basis for entering new specification details rather than full specifications.

\textsuperscript{7}The previous model is documented in [76].
The employees needed less time to understand the specifications and were not confused by irrelevant details.

- Inclusion of changes. In the case study, updates of viewspecs were propagated manually to the associated full specification. This was a rather tedious effort and associated computer support is needed.

For more comprehensive descriptions, the interested reader is referred to [68].

10.4 General Applicability

This section examines the general applicability of the suggested approach.

The idea of complexity reduction is not tailored to any particular modelling language or modelling approach. Although the idea is elaborated particularly for the TEMPORA languages and later on adapted to PPP language, it is not restricted to these languages. The framework for complexity reduction certainly applies to all approaches. For example, the notion of viewspecs, filters and abstraction levels is not tailored to any specific approach or language. When it comes to the specific structuring and filtering mechanisms suggested for the TEMPORA languages, the definitions need to be adapted to the specific constructs of a language such mechanisms should be defined for. However, the general definitions of mappings apply. The general applicability of the suggested approach to complexity reduction will be examined as follows:

- The coupling between PID and ERL.
- Other structuring mechanisms.
- Filtering mechanisms.

The coupling between the PID and the ERL

In principle, the coupling between PID and ERL as defined in Chapter 5 certainly applies to all approaches based on DFD languages and logic based rule languages. This together with the fact that DFD languages and logic based rule languages are well known and wide spread (cf., Chapter 2) ensure that this coupling may be applicable to a wide range of approaches. However, the coupling for a particular modelling approach may need to be modified due to the fact that the definitions of the coupling in Chapter 5 is based on the specific semantics of ERL. Thus, logic based approaches which differ with respect to the, e.g., temporal semantics need to be modified.
An example of the general applicability of the coupling is already demonstrated by
the implementation of the coupling in COMEX. As already mentioned in Chapter 2,
COMEX is a tool used for symbolic evaluation of rules to support knowledge acqui-
sition [86, 149]. An intermediate version of the coupling between the process based
and rule based approaches was used a basis for providing an overall structure of rules
for development of knowledge based systems.

Other Structuring Mechanisms

The framework for structuring mechanisms is language independent as well as appli-
cation/domain independent. It gives a formal basis for describing different types of
links 1) within a specification and 2) between specifications. The latter type of links
may involve links between components which may differ with respect to modelling
language and level of specification. The framework may be used to describe how
components of specifications are linked in any modelling approach irrespective of
languages and domains.

The specific structuring mechanisms for TEMPORA described in Chapter 6 are only
useful for approaches which are based on ER, DFD and/or logic. For instance, it
was very easy to adapt the structuring mechanisms for TEMPORA to PPP since
they are based on the languages from the same modelling orientations. However, any
discrepancies from semantics of the TEMPORA languages may cause changes. For
other approaches the specific structuring mechanisms suggested in this work may be
of little use.

Filtering Mechanisms

Similarly to the framework for structuring mechanisms, the framework for filtering
mechanisms is language independent as well as application/domain independent. It
gives a formal basis for describing abstraction levels, filters and views. The
formal definitions of filters can be applied to other languages by changing the data
structure and algorithms according to the new language’s semantics. This may not
involve too much work because:

- Many case environments apply similar conceptual languages (DFD based, ER
  based and/or rule based) and have similar problems when they are used for
  modelling of large and complex systems (information overload).

- The principle of defining appropriate filters may be generally applicable for
  visual languages as a complexity reducing technique since it is generally easy
to select relevant components from a diagram.

Some work has already been done in the student project described above [68], where
the major model filters defined in Chapter 7 were applied for some other languages,
e.g., how components filters can be used together with the OOA [24]. [68] also provides examples which show how language filters can be defined for, e.g., the ERAE [28], the approach by Wirfs-Brock [155] and Statecharts [52, 53].

10.5 Chapter Summary

This chapter has validated the suggested complexity reducing approach as follows: First, we have evaluated the results against the requirements in Chapter 4. The work suggested in this thesis provides additional support to develop large scale information systems following the PPP and TEMPORA approaches. In our work, complexity reducing facilities are embedded both as properties of the language and as properties of the methods and tools. The latter has been emphasised. Rather than extending the languages by defining new constructs, we have focused on the existing languages and provided additional support to improve their ability to deal with development of large systems. We have found that the perceptibility, expressive economy, method and tool potential and reducibility features of the TEMPORA and PPP languages have increased by the coupling between the process based and rule based approaches as well as the inter-language links. The expressive power has only been affected to a limited extent. As far as method and tool dependent requirements are concerned, we have mostly improved the structuring and viewing features. Moreover, we have found that the changeability features have increased and the modelling freedom has not been restricted, rather the opposite. This is achieved by providing the structuring and filtering mechanisms together with the complexity reducing approach as such which provides a way to develop specification using viewspecs as a basis.

Second, we have evaluated the results against other related approaches and found that our approach has some novelties compared to these. This applies to some of the parts suggested in this work as well as the approach as such. To our knowledge there are no other approaches that provide a formal definition of the coupling between a DFD based language and a logic based language including temporal aspects as provided in our approach. We have also provided the foundation for a flexible visualisation of rules through the coupling to the PID, through a set of rule relations and the possibility of viewing subsets of these relations. The basic idea of generating multiple viewspecs in our approach is quite similar to the way of generating views in some other approaches. Where our approach to filtering differs from the other approaches is in the scope and flexibility of viewspec generation. We define a wide range of different filters and if none of these are suitable, the user may define her own or even use a combination of manual and automated filters. Moreover, the use of viewspecs which are always related to a consistent system specification, together with the active use of the versioning of viewspecs in the modelling process are also novel to our approach.

Third, we have reported experiences using intermediate versions on examples and in case studies. Basic lessons learned from the experiences using the suggested facilities to complexity reducing features in case studies can be summarised as follows. Firstly,
the use of a complexity reducing approach seems promising. Secondly, the generation of viewspecs, its preparation for presentation and the inclusion of changes into full specifications based on viewspecs are tedious tasks and comprehensive tool support is therefore necessary to use viewspecs efficiently and effectively in the modelling process. Although we have done some experiences building specifications based on viewspecs more comprehensive field studies are necessary to test the suggested approach.

Finally, we have briefly discussed the general applicability of the approach and found that some of its features can also be applied to other modelling approaches. The idea of complexity reduction and the framework for complexity reduction can certainly be applied to other approaches. Moreover, the filtering mechanisms can also be adapted to be used together with other approaches. Although the least effort is to use the TEMPORA and PPP filters for similar languages, that is, ER-like, DFD-like and logic based languages, they can also be adapted to be used with other languages as has also been shown. The TEMPORA structuring mechanisms are only however useful for similar languages and must be completely redefined if other languages are to be used.
Chapter 11

Concluding Remarks and Future Directions

The chapter sums up the achievements of the work and mentions some aspects that should be dealt with more extensively, giving directions for further work.

11.1 Major Contributions

We have investigated complexity reduction in information modelling in order to deal with development of large and complex systems specifications. The dual role of specifications in information systems development has been examined. In particular, we have focused on how we can overcome the problems of fragmentation and complexity explosion. The result of our work is a set of complexity reducing facilities along with comprehensive methodological support. More specifically, the major contributions of our work are as follows:

- We have discussed different kinds of complexity reducing features in conceptual modelling: information hiding, structuring, viewing, translation and modification of layout. We have also examined how contemporary CASE tools and state-of-the-art approaches to conceptual modelling deal with large and complex systems. Although the approaches to conceptual modelling provide a wide range of complexity reducing features, the support for such features in contemporary CASE technology is poor.

- A framework for complexity reduction in information systems development has been developed. This includes definitions of basic concepts and frameworks in which structuring and filtering mechanisms can be explained. Few contemporary approaches have addressed the duality of specifications. A better understanding of complexity in specifications and ways to reduce the complexity are therefore important and useful. The framework for structuring a conceptual
model gives us means to describe how information should be structured within and across specification levels. This is crucial in order to reduce fragmentation in specifications. The framework for filtering mechanisms provides us with means to focus on essential issues in specifications by suppressing irrelevant details. The filtering mechanisms deal with information overload.

- We have developed a set of structuring and filtering mechanisms for the PPP and TEMPORA languages.

- Suggestions for how to deal with typical problems of rule based approaches in information systems development are provided, and the emphasis is on how these can be dealt with in PPP and TEMPORA:
  
  - Explicit representation of control structures by utilising the coupling to a diagrammatic modelling language. The coupling between process based languages and rule based languages is formally defined. Using this coupling, it is possible to go back and forth between process representations and the corresponding rule representations.
  
  - Structuring of rule bases (ways to represent and relate rules at different levels of abstraction).
  
  - Viewing of rule bases (ways of viewing a rule base from different perspectives).

- We have investigated the potential for viewpoint-oriented development in CASE environments. Our approach does not use a synthesis of viewspecs as a basis for generating the executable information system but it rather suggests viewspec as a means of assisting the construction of large system specifications. To deal with the problems of inconsistency among specifications, viewspecs are always associated with a consistent system specification. The versioning system developed in [3] is used to manage a large number of viewspecs and various versions of viewspecs in the modelling process. The system has however been further modified and extended to fit better with our complexity reducing approach.

- A discussion and suggestion for how complexity reducing facilities can be implemented in the PPP CASE tool, are provided (Chapter 9). This is used as a basis for the implementation of such facilities in the PPP prototype environment which is being developed. The inclusion of these facilities does not require dramatic changes to the current PPP implementation. The complexity reducing features in the TEMPORA capture tool were all implemented by SISU.

The feasibility of the approach has been illustrated as follows: In addition to outlining how some of the suggested features are implemented in the TEMPORA CASE tool (Chapters 5 to 8), we have suggested how the PPP CASE tool can be extended to deal with rules and to encompass complexity reducing features (Chapter 9). In addition, an evaluation with respect to stated requirements and a comparison with other approaches were provided (Chapter 10). This showed that our approach has
11.2 Limitations and Future Directions

The following issues should be addressed in future work:

- Complete the design specification of the tools and implement the suggested complexity reducing facilities in the revised version of the PPP environment. The inclusion of the complexity reducing features suggested in this work is already included in the PPP project plan.

- The suggested approach should be tested in field studies. They should focus on all aspects of the approach and investigate its usefulness in the various facets of the modelling process.

- Experiences from field studies may form the basis for development of more advanced filtering mechanisms. In particular, it would be interesting to investigate how user-specific filters can be developed based on user modelling and also how modification of layout can be supported without trading familiarity with automation.

- In this work, we have not addressed how we can provide support for updating of aggregates (e.g., complex objects) beyond support of manual inspection. Thus, updates of aggregates need further investigation.

- Specification integration is only briefly addressed in this work. We need to further investigate how changes can be propagated from updated viewspecs to the corresponding full specification in order to produce a new revised consistent version of the full specification. More comprehensive support of specification integration is necessary. In particular, it may be interesting to investigate how CSCW techniques can be applied to assist the merging process when a group of developers is involved.

- Finally, we would also like to extend our approach to deal with organisational aspects. Field studies together with existing viewpoint-oriented approaches such as ViewPoint-Oriented Development [38] may be central in this work.

Although much work is left to support all the activities of a complexity reducing approach, we have in this thesis demonstrated the importance of focusing of complexity reduction in information systems modelling.
Appendix A

Rule Relations

A set of relations were introduced to relate rules in a flat rule base [116, 156, 135]:

- **motivates** and **necessitates** describe hierarchies of purpose (Figure 7.35), i.e., correspond to goal hierarchies in organisation theory. They express the purpose of one rule in terms of one or more rules at a higher level, that is, explain why the organisation has a particular rule in terms of rules at the higher level. Given the rules A and B. A motivates B means that A is at a higher level than B and A gives the rationale behind B. By using the relation necessitates, an even stronger relationship between A and B can be expressed. A necessitates B indicates that A is at a higher level than B and the contents of B is necessary to achieve the contents of A. Thus, if A necessitates B we may also say that A motivates B (even though the opposite may not be true).

- **refers-to** and **causes** were introduced to form rule hierarchies based on definitions and causal matters. Two rules can be related by the refers-to relation if one rule depends on or contains a concept defined by the other rule. Given the rules A and B. A refers-to B means that A contains a concept which is defined by B. In order to compute A a computation of B might be necessary. The causes relation expresses a causal relationship between two rules.

- **overrules** and **suspends** were introduced to form rule hierarchies for explicit exception handling. Two rules can be related hierarchically if one rule overrules or suspends the other rule. The overrules relation deals with situations where one rule is given priority (in an absolute sense) to another rule. In the case of the suspends relation rule B would not have been discarded but considered for evaluation when rule A had finished its execution.

Experiences using the relations (e.g., motivates and necessitates relations) as defined above on practical problems showed that the major deficiency using such relations is that they are ambiguous. For example, what does it mean that a rule motivates another? Does it mean that the one rule motivated the capture of the other rule
or that one rule is consequence of the fact that the other rule holds? We may also
give other interpretations to these relations. Our experience is that each developer
seemed to have different opinions about the meaning of the relations because their
names are similar to terms often used (misused) in natural language. Thus, the same
problem as using quasi-natural language appears. People think they know what it
means because it is close to natural language but the intention of the relation was
quite often different from what the participants thought it would be. However,
the overall impression of providing such relations to model particular relationships
among rules were positive. They can be summarised as follows:

- **Different perspectives of a rule base.** The experiences indicate that there is a
  need to provide mechanisms to record various relationships between rules to
  allow for viewing the rule base from different perspectives. This will improve
  the comprehensibility of the rule specification by focusing on relevant parts
  of a specification using the relations. For instance, maintenance may benefit
  from this increase in comprehensibility.

- **Navigation support.** It is possible to navigate between specification components
  by using the relations. This also facilitates easy access from one specification
  to related components contained in other specifications (although this had to
  be done manually).

- **Vagueness versus preciseness.** The requirements to vagueness versus precise-
  ness of specifications differ in the different phases of development. At higher
  levels it may not be of any interest to model how all the various elements are
  linked since the number of cases may be too comprehensive¹. Modelling the
  high level rules may also cause problems because of organisational issues. For
  example, the formal rules may be different from the informal rules practiced
  in the business. Models at high levels of abstraction are however, vague by
  nature as reflected in the motivates and necessitates relationships. Since these
  may mean a lot of things there is trade-off between the vagueness and the pre-
  ciseness in order to provide useful information at a high level. At lower levels
  it may be easier to relate the rules since the domain knowledge is usually more
  precise.

- **Textual versus graphical representation of rule relationships.** At present the
  relationships between rules are only shown textually. This limits the usefulness
  of relations by the fact that it is rather awkward to get an overview of the rules
  that are related by the textual representation, i.e., low comprehensability. A
  graphical facility for showing structural relationships would be very useful for
  the modelling process, e.g., easy to see how various rules relate to each other.

- **Automated versus manual links.** If non-automated links are provided, there is
  a price to pay. The links and its associated features need to be entered and
  maintained if any changes to the information they connect occur. Thus, there
  is trade-off between structuring and non-structuring features associated with a

---
¹It may involve too many people to provide the necessary knowledge, or the knowledge may not
be available.
set of modelling languages. We want flexibility on one hand and changeability on the other. These do not necessarily go hand in hand in all situations. Experience has also shown that the use of links are different in different phases. In the early phases, manual links may be very useful but in the later stages when the information load is huge automated links seem to be the only solution. The specifications become simply too big to be maintained manually.

Although the overrules and suspend relations seemed to be promising for specifying exceptions as described in Chapter 3, they still remain to be found useful in practice. During the case studies we simply never found the need for them. If this was due to lack of means to capture them or the fact that they can be expressed differently [134], also remain to be found out. Further investigation is needed to determine if these relations are necessary or not. What can be stated however, is that they in any case should be limited to the specification level and thus they will not be reflected in the computational model.

The interested reader is referred to [116, 135] for a more detailed description of the rule relations.
Appendix B

Versioning and Configuration Management

We give a brief description of the main features of a system for versioning and configuration management of specifications which is developed by Andersen in a companion thesis [3]. There are a number problems related to version and configuration control. In this appendix we will limit ourselves to three of the most important problems:

- *Identification and control of versions and configuration management* addresses how components of an information system can be distinguished and how various versions of the components are allowed to co-exist.

- *Configuration management* addresses how components of an information system can be put together to form a particular configuration.

- *Development transactions* address how tasks should be organised in a development project.

Identification and Control of Versions

Versions of components (also called development object) can be distinguished by associating a set of attributes with each version. Some versions of a component may replace one another. These correspond to sequential versions and are called revisions. Other versions may exist in parallel and are called variants. Accordingly, two versions can be either revision related or variant related. The difference between two related versions are characterised by a set of elementary (atomic) changes. It is assumed that each version is developed based on a unique existing version. Each version is associated with a set of attributes to describe the particular features of each version such as date of creation, type of change and name of developer.
A version graph is introduced to show graphically the relationships between versions. Figure B.1a shows a PID specification excluding triggering flows. It is further extended with triggering flows in Figure B.1b. The latter diagram is therefore a revision of the former diagram. Figure B.1c describes the same system but the symbols for processes and ERT_views are replaced with circles and the traditional store symbol, respectively. The only difference between the two diagrams is the external representation of the concepts and they are thus variant related to each other. The version graph for the specifications in Figure B.1 is shown in Figure B.2.

![Figure B.1: A simple example modelled in DFD.](image)

![Figure B.2: A simple version graph.](image)

**Configuration Management**

The dependencies among components of a system are recorded in order to construct and reconstruct consistent configurations of the system. The number of potential configurations is typically huge for an industrial-sized application. The selection of a consistent configuration may be done in various ways such as freezing and recording interesting configurations or selecting configurations from intentional descriptions of their properties. The configuration is usually a time consuming process and the
optimisation is highly dependent on the granularity of the recorded dependencies among system components. The relationships between a system and its component specifications are depicted in a component structure graphs. Figure B.3 shows the component structure graph for a system modelled expressed using the TEMPORA language.

![Component Structure Graph](image)

**Figure B.3:** An example of a component structure graph for a system expressed using the TEMPORA language.

Each component of a system specification may exist in various versions as described above. To construct and reconstruct consistent configurations of such a specification, we distinguish between two types of specifications [3]:

- **Specifications expressed using hierarchical languages.** For hierarchical specifications a change in a low level specification (e.g., a PID diagram at the lowest level of specification) may cause a cascade of changes in versions of diagrams at higher level of decomposition. To avoid this effect, a system component is related to their decompositions by a solved by relation in a component structure graph. This decoupling of components and their decompositions makes it possible to delay changes until the developer explicitly wants to propagate them. The component structure graph for a PID specification which consists of three processes, P1, P2 and P3 where P2 is decomposed into three subprocesses, is shown in Figure B.4.

To achieve global name uniqueness for hierarchical specifications, the specification components' names are qualified with the name of the specification.

- **Specifications expressed using flat languages.** The concept of conceptual view [121] is introduced in the versioning system to deal with co-existing subsets of flat specifications. A new conceptual view can be built from existing conceptual views and is then related with a built from relation. Let us say that we have an ERT specification which is split into two separate specifications. Both of the specifications may then be extended with more complete information, updated because of previously detected errors, etc., that is, new revisions of each of the specifications may be developed by the same or different people.
specifications result. The component structure graph for the specification and its conceptual views is shown in Figure B.5. In a similar manner as with hierarchical specifications, global name uniqueness for flat specifications is achieved by qualifying the conceptual view components' names with the name of the conceptual view.

The Concept of Development Transaction

The concept of development transaction is central in organising tasks to be carried out in a development project. A development transaction is defined as the task of defining a new version of a development object\(^2\) [3]. In contrast to database

\(^2\)A development object may be a full specification, a viewspec or a component of a specification.
transactions\textsuperscript{3}, a development transaction is triggered, controlled and terminated by the developer. There are also other differences from database transactions \cite{72}: strict serialisability is not achievable, rollback is normally not feasible and update conflicts cannot generally be handled automatically.

Development transactions can be classified as \textit{long transactions} and \textit{nested transactions}. The former reflects that the duration of such a transaction may be long in the range of days and weeks. To manage such a transaction in a system, protocols or rules are developed to allocate resources, determine responsibility and provide work procedures to developers in a systematic manner. Otherwise developers might access and update other developers versions and a chaos would result. Nested transactions are introduced to reflect work task structures of development projects. Thus, a complex transaction may be divided into subtransactions to manage complex tasks.

Development transactions are implemented by the \textbf{check-out} and \textbf{check-in} operations (depicted in Figure B.6). The former creates a private repository or workspace of the development object that the transaction is going to work upon. The latter returns a new version of the development object, i.e., the result of the transaction, to the shared repository.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig_b6.png}
\caption{The check-out and check-in operations of a transaction.}
\end{figure}

\textsuperscript{3}Such transactions are found in traditional transaction oriented information systems. Their features are already briefly described in Chapter 5.
Appendix C

PPP Repository Structure

The repository structure of PPP developed by M. Yang is shown in Figure C.1 (adapted from [158]).

Figure C.1: The PPP repository structure.
Appendix D

Example of the Use of Rules in the PPM

To illustrate how we can use the prototype to generate executable code directly from the PPM and PLD diagrams, we will use an elementary banking example (already published in [67]).

Banking Example

The bank can open or close accounts for their clients. An account is identified by an account number and a client can hold one or more accounts. Furthermore, an account has to be opened before it can be closed or changed, otherwise an error message is to be issued by the system. The bank receives transactions in the form of debits and credits from their clients and the balance of the specified account is updated accordingly.

A statement is produced whenever a change of the balance of an account has occurred and in addition, a statement is issued at the end of each week for each account.

The development steps from the specification of the banking example at the analysis level to an executable specification follows the scenario outlined in the previous section:

Step 1: Develop PPM specification
The PPM specifications are modelled by PPM2001 as shown in Figures D.1, D.2 and D.3, respectively.

Step 2: Generate database schema from the Phenomenon model
From a subset of the phenomenon model (entities, objects, connections, and datatypes) a set of Prolog facts is generated. The flows to and from stores are linked to views of the static model and is in this way related to the Prolog facts.
Figure D.1: The phenomenon specification of the banking example.

Figure D.2: The process specification of the banking example.
Figure D.3: The PLD specification of process P4 in the banking example.

Step 3: Generate TEQUEL rules and C code
From PPM’s internal representation of the example, TEQUEL rules and C code are generated. TEQUEL rules are generated from the process structure whereas C code is produced from the PLD specification. Parts of the produced TEQUEL rules and the C code are shown in figure D.4 and figure D.5, respectively.

Step 4: Run the system
The specifications generated in step 3 are used as input to the temporal rule manager. Figure D.6 shows the execution\(^1\) of the specification during ticks 0 through 2.

\(^1\)The rule manager is run in verbose mode.
issuestatement(X1,X2)=action(
    dotrue: {call cissuestatement(X1,X2)}
);

issueweeklystatement(X1)=action(
    dotrue: {call cissuestatement(X1)}
);

rules

issuestatement(X1,X2) <=
    <+ updatedbalance(X1,X2).

issueweeklystatement(X1) <=
    <+ endofweek(X1).

openaccount(X1,X2,X3) <=
    <+ newcontract(X1,X2,X3).

Figure D.4: Some TEQUEL rules generated from the specification in step 1.
```c
pred_name=BIM_Prolog_string_to_atom("updatedaccount");
pred=BIM_Prolog_get_predicate(pred_name,2);
BIM_Prolog_call_predicate(pred,
  BPM_IN|BPS_SIMPLE|BPT_INTEGER,account
  ,BPM_IN|BPS_SIMPLE|BPT_INTEGER,amount-2
);

printf("%d",account);
printf("%s",BIM_Prolog_atom_to_string(name));
printf("%d",amount-2);
```

Figure D.5: C code fragments generated from the specification in step 1.

![Diagram](image_url)

Figure D.6: Running the system through the ticks 0, 1, and 2.
Appendix E

Rules - Sweden Post Case Study

A1.1 Event-action rules

R206111 Receive covering letter:

when covering_letter_and_delivery_notes(S,PO,ND,NDLs,HeadInfo,LineInfo)
if
    post_office.P 
    [has post_office_number.P0, 
     has latest_serial_number.PS 
    ] and
    PS=S-1 
then status("Serial number is correct") and 
    post_office.P has latest_serial_number.S and 
    Head member_of HeadInfo and Head == <{DN_No,_,_,_}> and 
    Lines member_of LineInfo and Lines == <{DN_No,_,_}> and 
    enter_dn_head(S,PO,ND,NDLs,Head,Lines)

R206112 faulty serial# covering letter:

when covering_letter_and_delivery_notes(S,PO,_,_,_,_)
if
    post_office [has post_office_number.P0, has latest_serial_number.PS] and 
    not PS = S-1 and CS = PS+1
then
    status("Expected serial number:" ,CS)

serial_no_ok Check if serial number is ok

if
    post_office [ 

283
has latest_serial_number = OldS, has post_office_number = P0
] and
S = OldS + 1
then
serial_no_ok (S,P0)

R206121 Register delivery note head - 1

when
enter_dn_head(S,P0,ND,NDLs,Head,Lines)
if
Head == <[DNo,DNt,<[A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,
A14,A15,CuN,CuNo,A16,A17,A18,A19]>> and
post_office.P has post_office_number.P0 and
agreement.A [has agreement_number.AgNo, valid_for agreement_customer.C
[ has customer_name.CuN, has customer_number.CuNo,
has_invoicing_address address.Adr [ contain street_and_number.A17,
contain zip_code.A18, contain city.A19]]] and
(address.Adr contain care_of.A16 or A16="var")
then
just_before always_in_future delivery_note.DN [ belongs_to customer.C, belongs_to latest_serial_number.S,
delivered_at post_office.P,
has total_postings_received [ has number_of_postal_items.A1, has number_of_parcel.A2 ],
has registry_clerk_signature.A3,
has sender [ has name.A4, has address [ contain care_of.A5, contain street_and_number.A6,
contain zip_code.A7, contain city.A8 ] ],
has delivery_note_no.DNo, has delivery_note_type.DNt,
has journal_name.A9, has journal_edition.A10,
has journal_number.A11, has partial_posting.A12,
has last_posting_major_postal_item.A13,
has administrative_officer [ has name.A14, has telephone.A15 ],
follows agreement.A ] and
enter_dn_head_ok(C,P0,DN,AgNo,ND,NDLs) and
<{ DNo,Line_Entries }>= Lines and Line member_of Line_Entries and
enter_dn_line(Line,C,P0,DN,AgNo,ND,NDLs)

R206122 Register delivery note head - 2

when
enter_dn_head(S,P0,ND,NDLs,Head,Lines)
if
Head == {DNo,DNo,<A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,
A14,A15,CuN,CuNo,A16,A17,A18,A19} and
post_office.P has post_office_number.P0 and not AgNo = "var" and
not agreement has agreement_number.AgNo and
customer.C [has customer_name.CuN, has customer_number.CuNo,
  has_invoicing_address address.Adr [
    contain street_and_number.A17,
    contain zip_code.A18, contain city.A19]] and
(address.Adr contain care_of.A16 or A16="var")
then
just_before always_in_future delivery_note.DN [ 
   belongs_to customer.C, belongs_to latest_serial_number.S,
   delivered_at post_office.P,
   has total_postings_received [ 
     has number_of_postal_items.A1, has number_of_parcel.A2 
   ],
   has registry_clerk_signature.A3,
   has sender [has name.A4, has address [ 
     contain care_of.A5, contain street_and_number.A6,
     contain zip_code.A7, contain city.A8 
   ]
   ],
   has delivery_note_no.DNo, has delivery_note_type.DNo,
   has journal_name.A9, has journal_edition.A10,
   has journal_number.A11, has partial_posting.A12,
   has last_posting_major_postal_item.A13,
   has administrative_officer [ 
     has name.A14, has telephone.A15 
   ]
] and
enter_dn_head_ok(C,P0,DN,AgNo,ND,NDLs) and
status("Delivery note is entered but the agreement number does not exist")
and {DNo,Line_Entries } == Lines and Line member_of Line_Entries and
enter_dn_line(Line,C,P0,DN,AgNo,ND,NDLs)

R206123 Reg DN head -3

when
modify_dn_head(S,P0,DNo,ND,NDLs,Head)
if
Head == {DNo,_,{A1,A2,A3,A4,A5,A6,A7,A8,A9,_,A10,A11,A12,A13,
A14,A15,CuN,CuNo,A16,A17,_,_} and
(delivery_note.DN has delivery_note_no.DNo or
 not delivery_note has delivery_note_no.DNo) and
(AgNo = "var" or not agreement has agreement_number.AgNo) and
customer.C [ has customer_number.CuNo, has customer_name.CuN]
then
delivery_note.DN [ 
   belongs_to customer.C, belongs_to latest_serial_number.S,
   delivered_at post_office.P0,
has total_postings_received [has number_of_postal_items.A1, has number_of_parcel.A2],
has registry_clerk_signature.A3,
has sender [
  has name.A4, has address [
    contain care_of.A5, contain street_and_number.A6,
    contain zip_code.A7, contain city.A8
  ]
],
has delivery_note_no.DNo, has delivery_note_type.A10,
has journal_name.A11, has journal_edition.A12,
has journal_number.A13, has partial_posting.A14,
has last_posting_major_postal_item.A15,
has administrative_officer [has name.A16, has telephone.A17]
] and status("No agreement number or a nonexistent one, was given") and enter_dn_head_ok(C,P0,DN,AgNo,ND,NDLs)

R206124 Register delivery note head - 4

when modify_dn_head(S,P0,DNo,ND,NDLs,Head)
if Head == <{DNo, , <{A1,A2,A3,A4,A5,A6,A7,A8,AgNo, , A10,A11,A12,A13, A14,A15,CuN,CuNo,A16,A17, , ,}>}> and (delivery_note.DN has delivery_note_no.DNo or not delivery_note has delivery_note_no.DNo) and not "var" = AgNo and agreement [has agreement_number.AgNo, valid_for agreement_customer.C [ has customer_number.CuNo, has customer_name.CuN]]
then delivery_note.DN [
  belongs_to customer.C, belongs_to latest_serial_number.S, delivered_at post_office.P0,
  has total_postings_received [has number_of_postal_items.A1, has number_of_parcel.A2],
  has registry_clerk_signature.A3,
  has sender [
    has name.A4, has address [
      contain care_of.A5, contain street_and_number.A6,
      contain zip_code.A7, contain city.A8
    ]
  ],
  has delivery_note_no.DNo, has delivery_note_type.A10,
  has journal_name.A11, has journal_edition.A12,
  has journal_number.A13, has partial_posting.A14,
  has last_posting_major_postal_item.A15,
  has administrative_officer [has name.A16, has telephone.A17]
] and enter_dn_head_ok(C,P0,DN,AgNo,ND,NDLs)
R206131 Enter delivery note line -1

when
   enter_dn_line(DNLine,C,PO,DN,AgN,ND,NDLs)
if
   DNLine == <$Q,W,T,...,$> and
   Q = "var" and not W = "var" and not T = 0 % check if variables
then
   delivery_note_line.DNL [  
   belongs_to delivery_note.DN, has weight.W,  
   has invoice_text.T  
] and
   enter_dn_line_ok(DNLDNLine,C,PO,DN,AgN,ND,NDLs)

R206132 Enter delivery note line - 2

when
   enter_dn_line(DNLine,C,PO,DN,AgN,ND,NDLs)
if
   DNLine == <$Q,W,T,...,$> and
   not Q = "var" and W = "var" and not T = 0 % check if variables
then
   delivery_note.DN contain delivery_note_line.DNL [  
   has quantity.Q,  
   has invoice_text.T  
] and
   enter_dn_line_ok(DNLDNLine,C,PO,DN,AgN,ND,NDLs)

R206133 Enter delivery note line -3
when
   enter_dn_line(DNLine,C,PO,DN,AgN,ND,NDLs)
if
   DNLine == <$Q,W,T,...,$> and
   not Q = "var" and not W = "var" and not T = 0 % check if variables
then
   delivery_note_line.DNL [  
   belongs_to delivery_note.DN,  
   has quantity.Q,  
   has weight.W,  
   has invoice_text.T  
] and
   enter_dn_line_ok(DNLDNLine,C,PO,DN,AgN,ND,NDLs)

R206134 Enter delivery note line -4
when
   enter_dn_line(DNLine,C,PO,DN,AgN,ND,NDLs)
if
   DNLine == <$Q,W,T,...,$> and
   not Q = "var" and not W = "var" and T = 0 % check if variables
then
   delivery_note_line.DNL [
belongs_to delivery_note.DN,
has quantity.Q,
has weight.W
] and
enter_dn_line_ok(DNL, DNLine, C, PO, DN, AgN, ND, NDLs)

R206135 Enter delivery note line -5

when
  enter_dn_line(DNLine, C, PO, DN, AgN, ND, NDLs)
if
  DNLine == <{Q, W, T, _, _, _}> and
  Q = "var" and not W = "var" and T = 0% check if variables
then
  delivery_note_line.DNL [
    belongs_to delivery_note.DN,
    has weight.W
  ] and
  enter_dn_line_ok(DNL, DNLine, C, PO, DN, AgN, ND, NDLs)

R206136 Enter delivery note line -6

when
  enter_dn_line(DNLine, C, PO, DN, AgN, ND, NDLs)
if
  DNLine == <{Q, W, T, _, _, _}> and
  not Q = "var" and W = "var" and T = 0% check if variables
then
  delivery_note_line.DNL [
    belongs_to delivery_note.DN,
    has quantity.Q
  ] and
  enter_dn_line_ok(DNL, DNLine, C, PO, DN, AgN, ND, NDLs)

R206211 Check delivery note head - 1

when
  enter_dn_head_ok(C, PO, DN, AgN, ND, NDLs)
if
  agreement_customer.C has agreement.A has agreement_number.AgN and
  not (customer.C has reason_for_credit_block and credit_agreement.A)
then
  recount_check_head(C, PO, DN, AgN, ND, NDLs) and
  status("Delivery note head is ok")

R206212 Check delivery note head - 2

when
  enter_dn_head_ok(C, _, _, AgN, _, _)
if
    agreement_customer.C has agreement.A and agreement_number.AgN and
customer.C has reason_for_credit_block.R and credit_agreement.A
then
    status("Customer has a credit block. Reason:", R)

R20621 Check delivery note line - 1

when
    enter_dn_line_ok(DNL,Line,C,P0,DN,AgN,ND,NDLs)
if
    Line == <{...,...},AN,AC,N}> and
delivery_note_article.X [has article_number.AN,
    accounted_as article_group [has account.AC, has name.N]
] and
AN \leq 750 and AN =\leq 751 and
not delivery_note.DN has delivery_note_type.17 and
not grouped_postal_item.X
then
delivery_note_article.X is_registered_on delivery_note_line.DNL and
recount_check_line(C,P0,DN,AgN,ND,NDLs) and
generate_article(DN,C,X)

R206222 Check delivery note line - 2

when
    enter_dn_line_ok(DNL,Line,C,P0,DN,AgN,ND,NDLs)
if
    Line == <{...,...},AN,AC,N}> and
    (AN = 750 or AN = 751) and
delivery_note.DN has delivery_note_type.17 and
delivery_note_article.X [has article_number = AN,
    accounted_as article_group [has account.AC, has name.N]
] and
not grouped_postal_item.X
then
delivery_note_article.X is_registered_on delivery_note_line.DNL and
recount_check_line(C,P0,DN,AgN,ND,NDLs) and
generate_article(DN,C,X)

R206223 Check delivery note line - 3

when
    enter_dn_line_ok(DNL,Line,C,P0,DN,AgN,ND,NDLs)
if
    Line == <{...,...},AN,AC,N}> and
    AN =\leq 750 and AN =\leq 751 and
grouprated_postal_item.X
[ has article_number.AN,
    accounted_as article_group
}
[ has account.AC, 
  has name.N 
 ] 
} and 
not agreement_customer.C has price_regulating_agreement 
contain price_regulation regulates article.X 
then 
delivery_note_article.X is_registered_on delivery_note_line.DNL and 
status("The customer has not specified a priceregulating agreement") 
and recount_check_line(C,PO,DN,AgN,ND,NDLs) 

R206224 Check delivery note line - 4 
when 
  enter_dn_line_ok(DNL,Line,C,PO,DN,AgN,ND,NDLs) 
if 
  Line == <_,_,AN,AC,N> and 
  agreement_customer.C has price_regulating_agreement 
  contain price_regulation_grouprating [ 
    regulates grouprated_postal_item.X [ 
      has article_number.AN, accounted_as article_group [ 
        has account.AC, has name.N 
      ] 
    ] 
  ] 
} and 
delivery_note.DN has delivery_note_type.15 
then 
delivery_note_article.X is_registered_on delivery_note_line.DNL and 
recount_check_line(C,PO,DN,AgN,ND,NDLs) and 
generate_article(DN,C,X) 

R206231 Automatic generation of dln line - 1 
when 
  generate_article(DN,C,A) 
if 
  agreement_customer.C has price_regulating_agreement 
  contain price_regulation regulates 
  automatically_generated_additional_article_postal_item.AA 
generated_by delivery_note_article.A 
then 
delivery_note.DN contain additional_delivery_note_line 
registers delivery_note_article.AA 

R206232 Automatic generation of dln line -2 
when 
  generate_article(_,C,A) 
if 
  not agreement_customer.C has price_regulating_agreement
contain price_regulation regulates
automatically_generated_additional_article_postal_item
generated_by delivery_note_article.A
then
true

R20631 Re-count check 1

when
recount_check_head(_,PO,DN,_,ND,_,)
if
post_office.P [has post_office_number.PO, has latest_serial_number.S] and
number_of_bagof(DN, delivery_note.DN delivered_at_post_office.P) = NoDN
then
status("Current head count is",NoDN,ND,S,PO)

R20632 Re-count check 2

when
recount_check_line(_,PO,DN,_,_,NDLs)
if
CEEDNo,No> member_of NDLs and
post_office.P [has post_office_number.PO, has latest_serial_number.S] and
number_of_bagof(DNL, (delivery_note.DN
[ delivered_at post_office.P,
contain delivery_note_line.DNL,
has delivery_note_no.DNo
] and
not additional_delivery_note_line.DNL)) = NoDNL
then
status("Current line count is",NoDNL,No,S,PO,DNo)

R20633 Re-count check 3

when
transactionAlright(S,PO)
if
bagof(<DN,C>, delivery_note.DN [belongs_to latest_serial_number.S,
delivered_at post_office has post_office_number.PO,
belongs_to customer.C]) = DNs
then
accepted_delivery_notes(DNs) and
<DN,C> member_of DNs and
not delivery_note.DN belongs_to latest_serial_number.S

total_number_dln_lines_ok Check the total number of dln lines

if
NDLs =number_of_bagof(DNL, delivery_note [
has delivery_note_no.DN_NO, contain delivery_note_line.DNL, 
  delivered_at post_office has post_office_number.PO 
  ] and 
  not additional_delivery_note_line.DNL) 
then 
  total_number_dln_lines_ok(MDLs,DN_NO,PO) 

% else status("Actual no. of dln lines and control info.is different") 

total_number_dln_ok Check if total number of dln is ok 

if 
  number_of bagof(DN, delivery_note.DN delivered_at post_office [ 
    has post_office_number.PO 
  ] 
  ) = ND 
then 
  total_number_dln_ok(ND,PO) 

A1.2 Constraint rules 

R4 dn reg grouprated -> price reg agr 

if customer.C responsible_for claim is_based_on delivery_note 
  contain delivery_note_line registers grouprated_postal_item.A 
then 
  agreement_customer.C has price_regulating_agreement contain 
  price_regulation_grouprated refers_to grouprated_postal_item.A 

R22 pr reg agr no gr rate post item 

if 
  price_regulation.X and 
  number_of{C for which price_regulation.X contained_in 
    price_regulating_agreement valid_for agreement_customer.C} = 1 and 
  price_regulation.X regulates article.A 
then 
  not grouprated_postal_item.A and 
  article.A has article_type =$"grouprated_postal_item"

A1.3 Derivation rules
R69 derive dnl price na w q g
if
delivery_note.DN contain delivery_note_line.DNL [ 
    has weight.W, registers grouped_postal_item.A [ 
        has_kilo nominal_price.QNP, has nominal_price.KNP],
    has quantity.Q] and
not delivery_note.DN follows price_regulating_agreement contain
price_regulation regulates article.A and
QNPQ = QNP*Q and KNPK = KNP*W and
P= minimum <{QNPQ,KNPK}>
then
delivery_note_line.DNL has price.P

R77 derive dnl price na w nq nq
if
delivery_note.DN contain delivery_note_line.DNL and
delivery_note_line.DNL [ has weight.W, 
    registers priced_article.A has nominal_price.NP] and
not delivery_note.DN follows price_regulating_agreement contain
price_regulation regulates article.A and
not (delivery_note_line.DNL has quantity) and
not (delivery_note_line.DNL registers grouped_postal_item.A) and
P = W * NP
then
delivery_note_line.DNL has price.P

A1.4 ERT diagram - Delivery Note

The ERT diagram for the Delivery Note is shown in Figure E.1.

Rule form - R206111

An example of a rule is shown in Figure E.2.
Figure E.1: ERT specification of Delivery Note.
Form: Business Rule

Rule Id: R206111
Name: Receive covering letter
Category: Event/Action

Defined by: rolf/anne helga
Def. Date: 930515
Rev. Date: 931122

Informal Business Rule in English:

When a covering letter and belonging delivery notes are received, the serial number of the post office where the covering letter is received, is checked. If the serial number is correct the delivery notes are registered; otherwise the registration process will be rejected.

The parameters ND, NDLs are only used by Re_count_check (P6.3) but to avoid prompting the user for this information each time recount check is invoked, these parameters are passed along.

Formal Business Rule:

```plaintext
when covering_letter_and_delivery_notes(S,PO,ND,NDLs,HeadInfo,LineInfo)
  if
    post_office.P
    [has post_office_number PO,
     has latest_serial_number PS
    ] and
    PS=3-1
  then status("Serial number is correct") and
    post_office.P has latest_serial_number.S and
    Head member_of HeadInfo and Head == <DN_No_,_> and
    Lines member_of LineInfo and Lines == <DN_No_> and
    enter_dh_head(S,PO,ND,NDLs,Head,Lines)
```

Associates to: 

<table>
<thead>
<tr>
<th>ERT Object Name</th>
<th>Rule/DT Id</th>
<th>Motivates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>latest_serial_number</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Motivated by: 

<table>
<thead>
<tr>
<th>ERT Object Name</th>
<th>Rule/DT Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>latest_serial_number</td>
<td></td>
</tr>
</tbody>
</table>

Refers to: 

<table>
<thead>
<tr>
<th>ERT Object Name</th>
<th>Necessitated by:</th>
<th>Necessitates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>post_office_number</td>
<td>Rule Id</td>
<td>serial_no_ok</td>
</tr>
<tr>
<td>latest_serial_number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>post_office</td>
<td></td>
<td></td>
</tr>
<tr>
<td>invoices customer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>enter_dh_head</td>
<td></td>
<td></td>
</tr>
<tr>
<td>covering_letter_and_delivery_notes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implements:

| Process Name       | |
|--------------------||
| check_serial_no    | |

Figure E.2: An example of an ERL rule.
Appendix F

Definition of Filters

PID Component Filter

Formally, a PID component filter $\mathcal{F}_{\text{component}}$ can be expressed as follows:

$$\mathcal{F}_{\text{component}}: V^*_{PID} \rightarrow V_{\text{component}}$$

- $V^*_{PID} = (\text{Processes}^*_{PID}, \text{ERT\_views}^*_{PID}, \text{Flows}^*_{PID}, \text{External\_agents}^*_{PID}, \text{Ports}^*_{PID}, \text{Timers}^*_{PID})$
- $V_{\text{component}} = (\text{Processes}_v, \text{ERT\_views}_v, \text{Flows}_v, \text{External\_agents}_v, \text{Ports}_v, \text{Timers}_v)$
- $V_{\text{component}} \subseteq V^*_{PID}$. This is abuse of notation as already described in Section 7.1, that is, $V_{\text{component}}$ corresponds to $V$.
- $V_{\text{component}} = \mathcal{N}(P, V^*_{PID}, N)$ where $P \in \text{Processes}$, $N$ is an integer and $N \geq 0$
- $\text{Flow} = (\text{Flow\_name}, \text{Origin}, \text{Destination})$
- $\text{Flows}$ is a set of $\text{Flow}$.

We must then define the neighbour function $\mathcal{N}(P, S, N)$:

$$\mathcal{N}(P, V, N) = (P, \emptyset, \emptyset, \emptyset, \emptyset, \text{process\_ports}(P, \text{Ports})), \text{ where } N = 0$$

$$\mathcal{N}(P, V, N) = (\text{Processes}_N, \text{ERT\_view}_N, \text{Flow}_N, \text{External\_agent}_N, \text{Ports}_N, \text{Timers}_N), \text{ where } N > 0 \land$$
\[ N \( P, V, N-1 \) = \langle \text{Process}_{N-1}, \text{ERT\_view}_{N-1}, \text{Flow}_{N-1}, \text{External\_agent}_{N-1}, \text{Ports}_{N-1}, \text{Timers}_{N-1} \rangle \wedge \]

\[ \text{Process}_{N} = \text{Process}_{N-1} \cup \{ X \mid \langle \_, X, X_{N-1} \rangle \in \text{Flows}_{N-1} \wedge \langle \_, X_{N-1}, X \rangle \in \text{Flows}_{N-1} \wedge X \in \text{Processes}_{PID}^* \wedge X_{N-1} \in \text{Process}_{N-1} \} \wedge \]

\[ \text{External\_agent}_{N} = \text{External\_agent}_{N-1} \cup \{ R \mid \langle \_, \text{Process}_{N-1}, R_{N-1} \rangle \in \text{Flows}_{N-1} \wedge \langle \_, R_{N-1}, \text{Process}_{N-1} \rangle \in \text{Flows}_{N-1} \wedge R_{N-1} \in \text{External\_agent}_{N-1} \} \wedge \]

\[ \text{Timer}_{N} = \text{Timer}_{N-1} \cup \{ S \mid \langle \_, \text{Process}_{N-1}, S_{N-1} \rangle \in \text{Flows}_{N-1} \wedge \langle \_, S_{N-1}, \text{Process}_{N-1} \rangle \in \text{Flows}_{N-1} \wedge S_{N-1} \in \text{Timer}_{N-1} \} \wedge \]

\[ \text{ERT\_view}_{N} = \text{ERT\_view}_{N-1} \cup \{ Y \mid \langle \_, \text{Process}_{N-1}, Y_{N-1} \rangle \in \text{Flows}_{N-1} \wedge \langle \_, Y_{N-1}, \text{Process}_{N-1} \rangle \in \text{Flows}_{N-1} \wedge Y_{N-1} \in \text{ERT\_view}_{N-1} \} \wedge \]

\[ \text{Flows}_{N} = \text{Flows}_{N-1} \cup \{ Z \mid Z \in \text{Flows}_{PID}^* \wedge Z = \langle O, D \rangle \wedge \langle O \in \text{Process}_{N} \vee O \in \text{ERT\_view}_{N} \vee O \in \text{External\_agent}_{N} \vee O \in \text{Timers}_{N} \vee D \in \text{Process}_{N} \vee D \in \text{ERT\_view}_{N} \vee D \in \text{External\_agent}_{N} \vee D \in \text{Timers}_{N} \} \}
\]

**ERT Filters**

**Inclusive timestamping filter** The inclusive timestamping filter can be formally defined as follows:

\[ \mathcal{F}_{\text{timestamping\_incl}}: V_{ERT}^* \rightarrow V_{\text{timestamping\_incl}} \]

where

- \[ V_{ERT}^* = \langle \text{Entity\_class}_{ERT}^*, \text{Relationship\_class}_{ERT}^*, \text{Is\_a\_relationship}, \text{Value\_class}, \text{Complex\_entity\_class}, \text{Cardinality\_constraint}_{ERT}^* \rangle \]
• \( V_{timestamping\_incl} = \langle A (Entity\_class), B (Relationship\_class), C (Is\_a\_relationship), 0, E (Complex\_entity\_class), Cardinality\_constraint_v \rangle \)

• \( Relationship\_class_v \subseteq Relationship\_class^*_ERT \)

• \( Relationship\_class = (Name\_relationship\_class, Component1, Component2) \)

• \( Relationship\_class_v \) is a set of \( Relationship\_class \)

• \( A (Entity\_class) = \{ X \mid X \in Entity\_class, D (X) = timestamped\_entity \lor \exists Y. Y \in Relationship\_class^*_ERT, D (Y) = timestamped\_relationship, (Y = ⟨⟨, X , ⟩⟩ \lor Y = ⟨⟨, , X⟩⟩) \}

• The function \( D (P) \) finds the type of an instance \( P \).

• \( B (X) = \{ X \mid X \in Relationship\_class^*_ERT, D (X) = timestamped\_relationship \} \)

• \( C (X) \) is the same function as \( A (E) \) defined for the inclusive generalisation filter.

• \( E (Complex\_entity\_class) = \{ X \mid X \in Complex\_entity\_class, D (X) = timestamped\_complex\_entity \lor \exists Y. Y \in Relationship\_class^*_ERT, D (Y) = timestamped\_relationship, (Y = ⟨⟨, X , ⟩⟩ \lor Y = ⟨⟨, , X⟩⟩) \}

• \( Cardinality\_constraint_v \subseteq Cardinality\_constraint^*_ERT \) where each element of \( Cardinality\_constraint_v \) is associated with an element of \( Relationship\_class_v \)

**ERT component filter**  Formally, an ERT component filter \( F_{component} \) can be expressed as follows:

\[ F_{component} (E, N): V^*_ERT \rightarrow V_{component} \]

where

• \( V^*_ERT = \langle Entity\_class, Relationship\_class, Is\_a\_relationship, Value\_class, Complex\_entity\_class, Cardinality\_constraint \rangle \)

• \( V_{component} = \langle Entity\_class_v, Relationship\_class_v, Is\_a\_relationship_v, Value\_class_v, Complex\_entity\_class_v, Cardinality\_constraint_v \rangle \)

• \( V_{component} \subseteq V^*_ERT \). This is again abuse of notation as already described in Section 7.1, that is, \( V_{component} \) corresponds to \( V \).
• \( V_{\text{component}} = \mathcal{N}(E, V_{ERT}^*, N) \), where
  \( E \in \text{Entity\_class} \), \( N \) is an integer and \( N \geq 0 \)

• \( \text{Relationship\_class} = (\text{Name\_relationship\_class, Component1, Component2}) \)

• \( \text{Relationship\_class}_v \) is a set of \( \text{Relationship\_class} \)

The neighbour function \( \mathcal{N}(E, S, N) \) is defined in a similar manner as for the PID component filter.
List of Figures

1.1 Duality of systems specifications. ........................................ 5
1.2 Complexity of specification is dependent upon the number of parts and interconnections. ........................................ 7
1.3 The major activities of the complexity reduction approach. ........ 8
1.4 DFD specification of the Sweden Post ordering and invoicing system. 11
1.5 DFD specification of the library system. .............................. 13
1.6 DFD specification of the oil processing system. .................... 14
1.7 DFD specification of the acquisition activity. ....................... 16
1.8 Process P4.6 and its immediate components. ......................... 17
1.9 Process P4.6 including ports and triggering flow. ................. 19
1.10 Specification of the acquisition activity including triggering flows and ports. ................................................. 20
1.11 A view of the acquisition activity where only control flow is shown. 21
1.12 A component view of P4.6 with the details of ports abstracted away. 22

2.1 Overview of complexity reducing techniques and how they relate to validation techniques. ........................................ 29
2.2 Data-oriented languages: a) Basic ER, b) Generalisation in ERC+, c) Aggregation in ERC+. ................................. 33
2.3 Summary view. .......................................................... 36
2.4 a) Exploding view of operator A, b) Annotation view for stream y. 36
2.5 Focused slices, showing timing and critical path. .................... 36
2.6 a) Encapsulation and b) Information-hiding. .......................... 37
2.7 Subject layer. .......................................................... 38
2.8 Subject, class-&-object, structure and service layers. ............... 39
2.9 An example using class and subsystem cards. a) Drawing class card, b) Drawing subsystem card. ................................. 40
2.10 An example using collaboration graphs. a) The detailed graph, b) A simplified graph, c) The overall system. ............... 40
2.11 An example using a class hierarchy graph and a venn diagram. a) Class hierarchy graph for Drawing element, b) Venn diagram for Drawing element. ............................................. 41
2.12 Creation of a high-level object-class view. .......................... 42
2.13 The concept of high-level views. .................................... 43
2.14 Viewpoint Resolution. ................................................. 45
2.15 The strategy for viewpoint analysis. ................................... 46
2.16 BNM for a payroll processing system. a) Lower level BNM, b) Higher level BNM. ................................................. 47
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.17</td>
<td>Hierarchy (XOR) in Statecharts. a) shows an XOR-free equivalent of b).</td>
<td>47</td>
</tr>
<tr>
<td>2.18</td>
<td>Orthogonality (AND) in Statecharts. a) shows the AND-free equivalent of b).</td>
<td>47</td>
</tr>
<tr>
<td>2.19</td>
<td>a) Zooming in, b) Zooming out.</td>
<td>48</td>
</tr>
<tr>
<td>2.20</td>
<td>Unclustering.</td>
<td>48</td>
</tr>
<tr>
<td>2.21</td>
<td>Conditional.</td>
<td>49</td>
</tr>
<tr>
<td>2.22</td>
<td>Selection.</td>
<td>49</td>
</tr>
<tr>
<td>2.23</td>
<td>Generalisation (is-included) and Association (is-in) in ERAE.</td>
<td>50</td>
</tr>
<tr>
<td>2.24</td>
<td>Contexts in ERAE.</td>
<td>50</td>
</tr>
<tr>
<td>2.25</td>
<td>A Gist specification.</td>
<td>51</td>
</tr>
<tr>
<td>2.26</td>
<td>ViewPoint-Oriented Development.</td>
<td>53</td>
</tr>
<tr>
<td>2.27</td>
<td>a) Hiconion for the IFIP example and b) shows a slightly zoomed-out Hiconion for the same system.</td>
<td>54</td>
</tr>
<tr>
<td>2.28</td>
<td>A flowchart and associated views.</td>
<td>56</td>
</tr>
<tr>
<td>2.29</td>
<td>An overview of the survey.</td>
<td>59</td>
</tr>
<tr>
<td>3.1</td>
<td>The TEMPORA implementation architecture.</td>
<td>69</td>
</tr>
<tr>
<td>3.2</td>
<td>ERT Model for oil processing design.</td>
<td>70</td>
</tr>
<tr>
<td>3.3</td>
<td>Top-level PID model for oil processing design.</td>
<td>71</td>
</tr>
<tr>
<td>3.4</td>
<td>Decomposed PID model for oil processing design including ports.</td>
<td>71</td>
</tr>
<tr>
<td>3.5</td>
<td>The port symbols of the PID.</td>
<td>72</td>
</tr>
<tr>
<td>3.6</td>
<td>A simple port example.</td>
<td>73</td>
</tr>
<tr>
<td>3.7</td>
<td>A small composite port example.</td>
<td>73</td>
</tr>
<tr>
<td>3.8</td>
<td>An example Process Hierarchy.</td>
<td>74</td>
</tr>
<tr>
<td>5.1</td>
<td>Relationship between the PID and the ERL.</td>
<td>95</td>
</tr>
<tr>
<td>5.2</td>
<td>The semantics of and ports expressed using the ERL.</td>
<td>96</td>
</tr>
<tr>
<td>5.3</td>
<td>The semantics of xor ports expressed using the ERL.</td>
<td>96</td>
</tr>
<tr>
<td>5.4</td>
<td>The semantics of or ports expressed using the ERL.</td>
<td>97</td>
</tr>
<tr>
<td>5.5</td>
<td>The semantics of the cond ports expressed using the ERL.</td>
<td>97</td>
</tr>
<tr>
<td>5.6</td>
<td>The semantics of the rep ports expressed using the ERL.</td>
<td>97</td>
</tr>
<tr>
<td>5.7</td>
<td>Example of a process and its corresponding ERL rules.</td>
<td>99</td>
</tr>
<tr>
<td>5.8</td>
<td>Example of a process with an XOR port.</td>
<td>99</td>
</tr>
<tr>
<td>5.9</td>
<td>Example of a process with an OR port.</td>
<td>100</td>
</tr>
<tr>
<td>5.10</td>
<td>Example of a decomposed process and its corresponding ERL rules.</td>
<td>100</td>
</tr>
<tr>
<td>5.11</td>
<td>Temporal structure to describe temporal information.</td>
<td>102</td>
</tr>
<tr>
<td>5.12</td>
<td>Relating rules and databases.</td>
<td>103</td>
</tr>
<tr>
<td>5.13</td>
<td>Two flows of time: event time and transaction time.</td>
<td>107</td>
</tr>
<tr>
<td>6.1</td>
<td>Structuring mechanisms ensure integrated specifications.</td>
<td>110</td>
</tr>
<tr>
<td>6.2</td>
<td>Possible types of links between specification components.</td>
<td>111</td>
</tr>
<tr>
<td>6.3</td>
<td>Structuring mechanisms in TEMPORA at the specification level.</td>
<td>115</td>
</tr>
<tr>
<td>6.4</td>
<td>Links between specification and execution level.</td>
<td>128</td>
</tr>
<tr>
<td>7.1</td>
<td>Framework for describing abstraction levels.</td>
<td>132</td>
</tr>
<tr>
<td>7.2</td>
<td>Examples of different types of viewspecs: a) an intra-language viewspec, b) an inter-language viewspec and c) an inter-level viewspec.</td>
<td>134</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.3</td>
<td>a) The viewspec is a projection of the originating specification whereas b) is an approximation of the originating specification.</td>
<td>138</td>
</tr>
<tr>
<td>7.4</td>
<td>PID specification of the acquisition activity.</td>
<td>141</td>
</tr>
<tr>
<td>7.5</td>
<td>PID specification where the ports are abstracted away.</td>
<td>141</td>
</tr>
<tr>
<td>7.6</td>
<td>The acquisition activity where regular ports are substituted by abstracted ports.</td>
<td>142</td>
</tr>
<tr>
<td>7.7</td>
<td>PID specification where the ports are removed.</td>
<td>143</td>
</tr>
<tr>
<td>7.8</td>
<td>PID specification of the acquisition activity where ports are removed.</td>
<td>144</td>
</tr>
<tr>
<td>7.9</td>
<td>a) Process P with n input flows, b) the input flows are abstracted into a composite flow, c) Process P with n output flows, and d) the output flows are abstracted into a composite flow.</td>
<td>145</td>
</tr>
<tr>
<td>7.10</td>
<td>a) n directed flows with the same origin and destination and b) the n flows are abstracted into a composite flow.</td>
<td>146</td>
</tr>
<tr>
<td>7.11</td>
<td>Different layout alternatives for abstracted bidirectional flows with the same origin and destination.</td>
<td>146</td>
</tr>
<tr>
<td>7.12</td>
<td>Splitting flows with different destination.</td>
<td>147</td>
</tr>
<tr>
<td>7.13</td>
<td>Collapsing flows with different origin.</td>
<td>147</td>
</tr>
<tr>
<td>7.14</td>
<td>Problems of parallel flow filters.</td>
<td>148</td>
</tr>
<tr>
<td>7.15</td>
<td>Different solutions to parallel flow filters.</td>
<td>149</td>
</tr>
<tr>
<td>7.16</td>
<td>A flow with three different destinations is split into three branches.</td>
<td>149</td>
</tr>
<tr>
<td>7.17</td>
<td>PID specification which shows different ways of collapsing flows.</td>
<td>150</td>
</tr>
<tr>
<td>7.18</td>
<td>PID specification where only control flows are shown.</td>
<td>151</td>
</tr>
<tr>
<td>7.19</td>
<td>PID specification where the ERT views are removed.</td>
<td>153</td>
</tr>
<tr>
<td>7.20</td>
<td>Process hierarchy.</td>
<td>155</td>
</tr>
<tr>
<td>7.21</td>
<td>Process P4.6 and associated information (PID symbols left out).</td>
<td>156</td>
</tr>
<tr>
<td>7.22</td>
<td>Process P4.6 and associated information.</td>
<td>157</td>
</tr>
<tr>
<td>7.23</td>
<td>ERT specification of parts of the acquisition system.</td>
<td>158</td>
</tr>
<tr>
<td>7.24</td>
<td>A more detailed specification of the objects Client and Bibliographic document.</td>
<td>158</td>
</tr>
<tr>
<td>7.25</td>
<td>Only entity classes and their generalisation hierarchies are kept in the diagram.</td>
<td>159</td>
</tr>
<tr>
<td>7.26</td>
<td>The value classes are removed from the diagram.</td>
<td>161</td>
</tr>
<tr>
<td>7.27</td>
<td>The timestamped information is removed from the diagram.</td>
<td>162</td>
</tr>
<tr>
<td>7.28</td>
<td>Only the timestamped information is kept in the diagram.</td>
<td>163</td>
</tr>
<tr>
<td>7.29</td>
<td>Only entity classes and their generalisation hierarchies are kept in the diagram.</td>
<td>164</td>
</tr>
<tr>
<td>7.30</td>
<td>All generalisation hierarchies are removed from the diagram.</td>
<td>165</td>
</tr>
<tr>
<td>7.31</td>
<td>The complex entity and value classes are removed from the diagram.</td>
<td>166</td>
</tr>
<tr>
<td>7.32</td>
<td>Derived information is removed from the diagram.</td>
<td>167</td>
</tr>
<tr>
<td>7.33</td>
<td>The cardinality constraints are removed from the diagram.</td>
<td>168</td>
</tr>
<tr>
<td>7.34</td>
<td>Client and immediately connected components.</td>
<td>169</td>
</tr>
<tr>
<td>7.35</td>
<td>An example hierarchy of purpose: the motivates and necessitates relations.</td>
<td>170</td>
</tr>
<tr>
<td>8.1</td>
<td>The approach based on systematic complexity reduction.</td>
<td>174</td>
</tr>
<tr>
<td>8.2</td>
<td>PID specification of the Sweden Post ordering and invoicing system.</td>
<td>175</td>
</tr>
<tr>
<td>8.3</td>
<td>ERT model of the main concepts of the ordering and invoicing system.</td>
<td>176</td>
</tr>
<tr>
<td>8.4</td>
<td>The decomposition of P6 Register delivery note.</td>
<td>177</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.5</td>
<td>PID specification of P6.1.</td>
<td>177</td>
</tr>
<tr>
<td>8.6</td>
<td>PID specification of P6.2.</td>
<td>178</td>
</tr>
<tr>
<td>8.7</td>
<td>An example of a composite filter.</td>
<td>182</td>
</tr>
<tr>
<td>8.8</td>
<td>An overview of viewspec generation.</td>
<td>183</td>
</tr>
<tr>
<td>8.9</td>
<td>Different filters may be used to generate the same viewspec.</td>
<td>184</td>
</tr>
<tr>
<td>8.10</td>
<td>The same specifications may be used to generate different viewspecs.</td>
<td>184</td>
</tr>
<tr>
<td>8.11</td>
<td>Filtering across specification boundaries.</td>
<td>186</td>
</tr>
<tr>
<td>8.12</td>
<td>Viewspec that shows P6.1.2 Enter Delivery Note Head and its immediately connected components.</td>
<td>186</td>
</tr>
<tr>
<td>8.13</td>
<td>The relevant objects associated with P6.1.2 Enter Delivery Note Head.</td>
<td>187</td>
</tr>
<tr>
<td>8.14</td>
<td>Delivery note and its immediately connected subcomponents.</td>
<td>188</td>
</tr>
<tr>
<td>8.15</td>
<td>Agreement and its immediate subcomponents.</td>
<td>188</td>
</tr>
<tr>
<td>8.16</td>
<td>Generalisation hierarchies for Customer and Agreement.</td>
<td>188</td>
</tr>
<tr>
<td>8.17</td>
<td>Overview of the event-action rules associated with process P6.1, P6.2 and P6.3.</td>
<td>189</td>
</tr>
<tr>
<td>8.18</td>
<td>a) shows the event-action rules associated with P6.1.2 Enter Delivery Note Head and b) shows the rules associated with the entity class Customer.</td>
<td>190</td>
</tr>
<tr>
<td>8.19</td>
<td>caused-by relations, filled circles denote communication with timer, ERT_views or external agents.</td>
<td>190</td>
</tr>
<tr>
<td>8.20</td>
<td>a) shows has_information_derived_by and b) shows constrained_by relations.</td>
<td>191</td>
</tr>
<tr>
<td>8.21</td>
<td>PID specification of P6.1.2 including ports.</td>
<td>195</td>
</tr>
<tr>
<td>8.22</td>
<td>Viewspecs are filter related to their originating specifications.</td>
<td>197</td>
</tr>
<tr>
<td>8.23</td>
<td>The component structure graph for the ERT and PID specifications.</td>
<td>198</td>
</tr>
<tr>
<td>8.24</td>
<td>a) The ERT specification and associated viewspecs b) the PID specification and associated viewspecs.</td>
<td>198</td>
</tr>
<tr>
<td>8.25</td>
<td>Viewspecs are variant related to their originating specifications.</td>
<td>199</td>
</tr>
<tr>
<td>8.26</td>
<td>Viewspecs are context related to their originating specifications.</td>
<td>200</td>
</tr>
<tr>
<td>8.27</td>
<td>Local workspaces.</td>
<td>201</td>
</tr>
<tr>
<td>8.28</td>
<td>Revisions of specifications.</td>
<td>203</td>
</tr>
<tr>
<td>8.29</td>
<td>Revision of a PID viewspec.</td>
<td>203</td>
</tr>
<tr>
<td>8.30</td>
<td>Resolving the conflict immediately after a viewspec is updated.</td>
<td>204</td>
</tr>
<tr>
<td>8.31</td>
<td>Building a new revision based on updated viewspecs.</td>
<td>205</td>
</tr>
<tr>
<td>8.32</td>
<td>The merge relations.</td>
<td>207</td>
</tr>
<tr>
<td>9.1</td>
<td>An example of a PhM model.</td>
<td>214</td>
</tr>
<tr>
<td>9.2</td>
<td>Top-level PrM model for oil processing design.</td>
<td>216</td>
</tr>
<tr>
<td>9.3</td>
<td>Decomposed PrM model for oil processing design including ports.</td>
<td>216</td>
</tr>
<tr>
<td>9.4</td>
<td>An example of a PLD specification.</td>
<td>217</td>
</tr>
<tr>
<td>9.5</td>
<td>The implementation architecture of the PPP CASE tool.</td>
<td>218</td>
</tr>
<tr>
<td>9.6</td>
<td>PPP architecture where CRF is included as a separate module.</td>
<td>222</td>
</tr>
<tr>
<td>9.7</td>
<td>The rule editor.</td>
<td>224</td>
</tr>
<tr>
<td>9.8</td>
<td>Updated repository structure that incorporates the use of rules in the PPP languages.</td>
<td>225</td>
</tr>
<tr>
<td>9.9</td>
<td>The coupling between PPM and TEQUEL.</td>
<td>225</td>
</tr>
<tr>
<td>9.10</td>
<td>An example of a PhM model where all the attributes are removed.</td>
<td>227</td>
</tr>
<tr>
<td>9.11</td>
<td>The definition of a versioned entity.</td>
<td>229</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>9.12</td>
<td>Updated repository structure that incorporate versioning of viewspecs</td>
<td>229</td>
</tr>
<tr>
<td>9.13</td>
<td>Procedure calls from other components requesting services from the</td>
<td>230</td>
</tr>
<tr>
<td></td>
<td>Viewing Facility</td>
<td></td>
</tr>
<tr>
<td>9.14</td>
<td>Refined PPP and CWM (cooperative work management) architecture</td>
<td>231</td>
</tr>
<tr>
<td>9.15</td>
<td>Functions to support complexity reduction in the PhM editor</td>
<td>232</td>
</tr>
<tr>
<td>9.16</td>
<td>Performing filtering of a PhM specification</td>
<td>233</td>
</tr>
<tr>
<td>9.17</td>
<td>A viewspec where all attributes are removed</td>
<td>233</td>
</tr>
<tr>
<td>9.18</td>
<td>Specification of a composed filter in the PrM editor</td>
<td>234</td>
</tr>
<tr>
<td>9.19</td>
<td>A viewspec where ports are replaced by abstracted ports and all</td>
<td>234</td>
</tr>
<tr>
<td></td>
<td>non-triggering flows are removed</td>
<td></td>
</tr>
<tr>
<td>9.20</td>
<td>An ERL specification shown together with an overview of the current</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>workspace and a context related viewspec</td>
<td></td>
</tr>
<tr>
<td>10.1</td>
<td>Complexity reducing features in the TEMPORA capture tool</td>
<td>243</td>
</tr>
<tr>
<td>B.1</td>
<td>A simple example modelled in DFD</td>
<td>270</td>
</tr>
<tr>
<td>B.2</td>
<td>A simple version graph</td>
<td>270</td>
</tr>
<tr>
<td>B.3</td>
<td>An example of a component structure graph for a system expressed</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>using the TEMPORA language</td>
<td></td>
</tr>
<tr>
<td>B.4</td>
<td>An example of a component structure graph for specifications expressed</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>using a hierarchical language</td>
<td></td>
</tr>
<tr>
<td>B.5</td>
<td>An example of a component structure graph for specifications expressed</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>using a flat language</td>
<td></td>
</tr>
<tr>
<td>B.6</td>
<td>The check-out and check-in operations of a transaction</td>
<td>273</td>
</tr>
<tr>
<td>C.1</td>
<td>The PPP repository structure</td>
<td>276</td>
</tr>
<tr>
<td>D.1</td>
<td>The phenomenon specification of the banking example</td>
<td>278</td>
</tr>
<tr>
<td>D.2</td>
<td>The process specification of the banking example</td>
<td>278</td>
</tr>
<tr>
<td>D.3</td>
<td>The PLD specification of process P4 in the banking example</td>
<td>279</td>
</tr>
<tr>
<td>D.4</td>
<td>Some TEQUEL rules generated from the specification in step 1</td>
<td>280</td>
</tr>
<tr>
<td>D.5</td>
<td>C code fragments generated from the specification in step 1</td>
<td>281</td>
</tr>
<tr>
<td>D.6</td>
<td>Running the system through the ticks 0, 1, and 2</td>
<td>281</td>
</tr>
<tr>
<td>E.1</td>
<td>ERT specification of Delivery Note</td>
<td>294</td>
</tr>
<tr>
<td>E.2</td>
<td>An example of an ERL rule</td>
<td>295</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Complexity reducing features in modelling approaches. ................. 31
2.2 Complexity reducing features in modelling approaches (cont.). ....... 32
6.1 Overview of rule relations. .................................................. 116
7.1 An overview of filters in TEMPORA. ..................................... 139
8.1 Features of pre-defined and user-defined filters. .......................... 181
10.1 Complexity reducing features in PPP and TEMPORA approaches. 238
10.2 Evaluation of complexity reducing features in modelling approaches. 247
Bibliography


